سلسلة كن أسدا للإبداع

BE LION CREATIVITY

الشرح الوافي لتعلم لغة SQL من نبعها الصافي

خالد السعداني



الشرح الوافي
لتعلم لغة SQL
من نبعها الصافي

من إعداد : خالد السعداني



"يا أيها الذين آمنوا اتقوا اللّه و قولوا قولا سديدا. يصلح لكم أعمالكم و يغفر لكم ذنوبكم ومن يطع اللّه و رسوله فقد فاز فوزا عظيما"

الأحزاب : 70 و 71



قبل أن نبدأ

الحمد لله ابتداء وانتهاء، والصلاة والسلام على حبيبنا محمد صلى الله عليه وعلى آله وصحبه وسلم تسليما كثيرا وبعد:

فإنه من غير اللائق بأمة طالما حملت مشاعل التقدم والتطور أن تتخلف عن ركب التكنولوجيا، وأن تنشغل بتوافه الأمور من قبيل ما اصطلح عليه بالفن والإبداع وهو لعمري عن الفن بعيد بُعْدَ الشرق عن الغرب، حيث صرنا نرى كل الدعم يوجه إلى تشجيع أهل الغناء والرقص والكوميديا والأجدر أن تنفق هذه الأموال فيما يعود على الأمة بالنفع والصلاح لتستعيد مثابتها في الريادة والقيادة، ولن يتأتى ذلك بعد النجوم ولا بانتظار هطول الذهب، فإن الله جل وعلا لا يغير ما بقوم حتى يغيروا ما بأنفسهم.

علينا كأمة طامحة لفرض ذاتها أن تعمل ﴿ وَقُلِ اعْمَلُوا فَسَيَرَى اللّهُ عَمَلَكُمْ وَرَسُولُهُ وَالْمُؤْمِنُونَ (التوبة:105)﴾ في سبيل قيام نهضة علمية. قد يقول قائل بأن كل ما نورده الآن هو موجود وقد سبقنا إليه الغرب، أقول ليس عيبا أن نبدأ من حيث انتهى غيرنا، فهم أيضا بنوا حضاراتهم على أطلال حضاراتنا وقد ترجموا من أمهات الكتب العلمية ماالله به عليم، ناهيك عن تزوير الحقائق الذي قاموا به من نسب بعض الاكتشافات إلى مغموريهم على حساب فطاحلنا.



طبعا هذه المقدمة بعيدة كل البعد عن لغة SQL ولكن لابد من هذا التمهيد حتى تتحمس – عزيزي القارئ – إلى الإبداع في هذا المجال وألا تكتفي فقط بالأخذ مما جاء في غضون الكتاب بل عليك الاجتهاد واستغلال ملكة التفكير.

ينقسم كتاب "الشرح الوافي، لتعلم لغة SQL من نبعها الصافي" إلى خمسة فصول رئيسية. الفصل الأول والثاني يعرضان مجموعة من المعلومات والتقنيات النظرية والرياضية، بينما تقوم الفصول الثلاثة الأخيرة بعرض معلومات عملية تطبيقية، تدفعك إلى اكتساب تقنيات جديدة تخول لك احتراف لغة SQL مع برنامج SQL Server.

ولا ننسى كما جرت العادة، إلى تنبيهكم بأن لكل شيء إذا ما تم نقصان، ومجهودنا هذا مجهود بشري قد يشوبه النقص والخلل عن سهو أو عن سوء فهم، لذا نرجو من حضراتكم إعلامنا في حال وجود أي شائبة أو خطأ في هذا الكتاب.

نسأل الله عز وجل أن يجعل هذا العمل خالصا لوجهه الكريم، وألا يجعل للنفس ولا للهوى حظا فيه، كما نسأله عز وجل أن ينفع كل قارىء لهذا الكتاب ولو بالنزر القليل.

دام لكم البشر والفرح وبالتوفيق والسداد إن شاء الله.

خالد السعداني



لمن يوجه هذا الكتاب؟

هذا الكتاب موجه إلى كل المستويات:

- إن كنت **مبتدئا** سينطلق بك منذ البداية.
- إن كنت **متوسطا**، فهذه فرصتك لتتحقق من مكتسباتك ولتتعلم أشياء جديدة.
- إن كنت **محترفا**، فقد تحتاج هذا الكتاب كمرجع عربي مبسط تعود إليه كلما نسيت مفهوما ما.
- كل أمثلة الكتاب جربتها على برنامج إدارة قواعد البيانات SQL Server بنسختيه 2005 و 2008، بالنسبة لمستخدمي برنامج الأوراكل Oracle أو غيره فالمفاهيم الواردة في الكتاب مفيدة لكم أيضا سيما فيما يتعلق بالجانب النظري (الجبر التجريدي، كائنات قواعد البيانات، ...)

للتواصل المباشر مع صاحب الكتاب، التحقوا بنا على صفحة خطوة إلى الأمام:

https://www.facebook.com/Khotwa.Amam





ملحوظة

كل الأكواد التي يتضمنها الكتاب شغالة 100%، ولكن الخطأ والسهو وارد، لذا لو حدث معكم أي خطأ في تجريب الأوامر الواردة في الكتاب فالتمسوا لي المعاذير وراسلوني بها على بريدي الالكتروني جازاكم الله خيرا:

Khalid_ESSAADANI@Hotmail.Fr



كتب للمؤلف

سبيلك المختصر لتعلم لغة السي# - الأساسيات

http://www.kutub.info/library/book/7076

سبيلك المختصر لتعلم لغة السي# - برمجة الواجهات

http://www.kutub.info/library/book/11495

سلسلة خطوة إلى الأمام مع الفيجوال بزيك - الخطوة الأولى

http://www.kutub.info/library/book/8050

سلسلة خطوة إلى الأمام مع الفيجوال بزيك - الخطوة الثانية

http://www.kutub.info/library/book/10564

مدخل إلى xml **وتوابعه**(DTD, XSL,CSS)

http://www.kutub.info/library/book/8305

مدخل إلى الداتا أكسيس لاير في السي#

http://www.kutub.info/library/book/7576

تحزيم البرامج خادم / عميل في الفيجوال استوديو (نسخة فرنسية)

http://www.mediafire.com/?by3e3u4d1emvxgo



الفهرس

الفصل الأول: عموميات حول البيانات



الفصل الثاني : الجبر التجريدي



الفصل الثالث: مدخل إلى لغة SQL


BE LION
arabic formation


BE LION
arabic formation


BE LION
arabic formation


BE LION
arabic formation



الفصل الرابع: تطبيقات الجبر التجريدي في لغة SQL



الفصل الخامس: البرمجة في Transact SQL


BE LION
arabic formation



بسم اللّه على بركة اللّه



الفصل الأول

عموميات حول البيانات



عموميات حول البيانات

الملفات *Files*

الملف File **هو مجموعة من البيانات التي تنتمي إلى نفس النوع، وتنقسم الملفات إلى نوعين:**

- **الملفات النصية** Text File**: ويكون محتوى الملف عبارة عن بيانات نصية.**
- **الملفات الثنائية** Binary File**: تكون على شكل بيانات ثنائية** Binary Data**، وهذا النوع من الملفات يستخدم غالبا من قبل لغات البرمجة.**

تخزين البيانات

يسعى الإنسان دائما إلى تسهيل المهام عليه وتيسير كل عقبات الحياة، فلو نظرنا إلى أول إصدارات الحواسيب لوجدنا مساحات التخزين لديها صغيرة جدا، ناهيك عن بطء الوصول إلى البيانات بسبب ضعف أداء الحاسوب من جهة، وبسبب رداءة نظام التشغيل من جانب آخر، ولكن الإنسان بسبب ملكته الإبداعية فإنه طور ومازال يطور أداء الحاسوب آليا وبرمجيا، حتى حصلنا على حواسيب بكفاءات عالية وبطرق فعالة وسريعة لحفظ البيانات ولاستغلالها.



كانت بداية عهد الإنسان بتخزين البيانات في سنة 1956، حينما قام باختراع القرص الصلب Hard Disk (حسب موسوعة ويكيبيديا)، منذ ذلك العهد والإنسان يطور وسائل تخزين البيانات إلى يومنا هذا.

بالنسبة لطرق تخزين البيانات فإن أشهرها:

التخزين المباشر:

ويكون حفظ البيانات على شكل أسطر متتالية في ملفات، ويتميز هذا النوع من التخزين ببساطته وسهولته، ولكنه يبقى ضعيفا بسبب صعوبة استخراج البيانات منه لأنه ليست هنالك طريقة لجلب البيانات منه إلا من خلال رتبة السطر، إضافة إلى عيب آخر وهو أنه يأخذ حجما كبيرا، وهذا مثال لهذا النوع من التخزين:

```
Abu Bakr  ASSIDIQ
 Ômar Ibn ALKHATTAB
Ôtman Ibn AFFAN
 Ali Ibn ABI TALIB
```

التخزين التسلسلي:

تتم عملية التخزين بشكل متسلسل، بحيث كل سطر ينتهي بفاصل (غالبا الفاصلة العادية) ثم بعد ذلك يليه السطر الثاني على الشكل التالي:

```
Abu Bakr  ASSIDIQ , Ômar Ibn ALKHATTAB , Ôtman Ibn AFFAN , Ali Ibn ABI TALIB
```



هذا النوع من التخزين يتميز عن التخزين المباشر بكونه لا يأخذ حجما كبيرا، ولكنه لا يختلف عنه في طريقة البحث عن البيانات بحيث يجب المرور على كل الأسطر من البداية إلى غاية العثور على السطر المنشود.

التخزين التسلسلي المفهرس:

نفس طريقة التخزين السابقة، ولكننا نقوم بفهرسة للبيانات المخزنة في الملف، مثلا لو عندنا ملف لحفظ بيانات العمال (نقوم بحفظ الاسم، السن، العنوان مثلا)، فكل عامل يأخذ رقما ترتيبيا، وذلك بغرض تسريع وثيرة الوصول إلى العامل المبحوث عنه، لأن البحث لا يشمل البيانات وإنما يخص فقط فهارسها Index، لكن تبقى مسألة مراجعة فهارس البيانات صعبة لأنه من الواجب تحديثها عند كل عملية إضافة أو تعديل أو حذف.

عيوب طرق التخزين السابقة:

من عيوب التخزين المباشر والتسلسلي، أنه ليس هنالك ترابط وعلاقات بين الملفات، مثلا لو عندنا ملف يخزن بيانات الأستاذ، وملف يخزن قائمة التخصصات فمن المستحيل التواصل بينهما لأنهما ملفان منعزلان.

ومن جهة أخرى مسألة حماية البيانات فهي غائبة، فقد تقوم بحذف تخصص معين من جدول التخصصات، ولهذا التخصص بيانات في ملف الأساتذة فتكون هنالك بعثرة وخلط للبيانات، أما إذا كان الملف مشتركا في شبكة محلية فهنالك مشكلة كبيرة وهي تحديث



البيانات، فقد يشتغل مجموعة من المستخدمين على نفس البيانات مما يؤدي إلى خلل في حفظها، لهذا ستجد استعمال هذا النوع من تخزين البيانات مقتصرا على التطبيقات الصغيرة.

قواعد البيانات التراتبية:

في هذا النوع من أنواع تخزين البيانات، نتخلص من مشاكل الحماية وأيضا من مشاكل الربط بين الملفات، ولكن هنالك مشكل آخر...

قامت كل شركة منتجة لبرنامج لإدارة قواعد البيانات بتخزين البيانات على شكل قواعد بيانات تراتبية بنمط يخصها، وبالتالي أضحى من الصعب الإحاطة بكل برامج إدارة قواعد البيانات، لأن كل برنامج له طريقته الخاصة.

للإشارة فتاريخ ظهور هذا النوع من التخزين كان سنة 1960 حسب موسوعة ويكيبيديا.

قواعد البيانات الترابطية أو العلائقية:

أتى هذا النوع من أنواع تخزين البيانات لحل كل المشاكل السابقة، بحيث يتوفر على حماية عالية للبيانات، بالإضافة إلى إمكانيات ربط البيانات فيما بينها على شكل علاقات سنفصلها فيما بعد إن شاء الله، والميزة الباهرة التي أتى بها هذا النوع من التخزين هو اعتماد كل أنظمة إدارة قواعد البيانات العلائقية على لغة موحدة، أتدرن ماهي هذه اللغة؟؟ إنها لغة SQL.



في قواعد البيانات العلائقية يتم تخزين البيانات في جداول ثنائية البعد (تتكون من أسطر وأعمدة).

ماهي قاعدة البيانات ؟

هي مجموعة من البيانات المخزنة بشكل منظم، وهي من أهم الدعائم التي تقوم عليها المعلوميات، حيث من خلال قواعد البيانات نستطيع حفظ وتعديل وحذف المعلومات بطرق سلسة، وكذلك تتيح لنا استخراج البيانات المحفوظة كما نريد.

ماهو نظام إدارة قواعد البيانات ؟

هو برنامج خاص بإدارة قواعد البيانات، ويسمح لك بإضافة وتعديل وحذف البيانات عبر واجهات ونوافذ، أيضا من خلال ربط قاعدة البيانات بإحدى لغات البرمجة، ويتيح لك أيضا مشاركة قواعد البيانات داخل شبكة Network، وتوجد الآن العديد من أنظمة إدارة قواعد البيانات ولعل أشهرها (Microsoft ACCESS, Microsoft SQL Server, Oracle Database , MySQL, PostqreSQL, Sybase, IBM DB2, ..)

الأنموذج العلائقي Relationnal Model

ظهر هذا النظام على يد Edgard Franck Codd سنة 1970، وينبني هذا النموذج على مفهوم الجبر العلائقي (مفهوم رياضي) الذي يتيح الاستعلام عن البيانات الموجودة في نظام



يضم وحدات مترابطة فيما بينها، لذا يعد الجبر العلائقي بمثابة الجانب التنظيري للغة SQL حيث أن فهم الجبر العلائقي يساعدك على استيعاب هذه اللغة بكل سهولة.

نتج عن استخدام هذا الأنموذج العلائقي تنظيم البيانات بالطريقة المشهورة حاليا على شكل جداول Tables لتسهيل عمليات الإضافة والتعديل والحذف والعرض للمستخدم، مع الإشارة إلى أن البيانات مازالت تخزن في ملفات Files ولكن طريقة عرضها صارت أسهل وأيسر.



الفصل الثاني

الجبر التجريدي



الجبر العلائقي Relationnal Algebra

عبارة عن مفهوم رياضي محض، أعتقد لو كنت من هواة الرياضيات سيكون قد مر على مسامعك من دون شك، وهو يقوم على أطروحة المجموعات Group Theory**، والغاية منه هو الحصول على بيانات جديدة من خلال بعض العمليات التي نقوم بها على وحدات (جداول) أخرى، وهذه العمليات هي:**

العمليات التجميعية :

تطبق هذه العمليات على مجموعتين.

الاتحاد Union (Ć):

R1 R2

الاتحاد Union **هو علاقة تربط بين مجموعتين لهما نفس الحقول ونفس الخصائص، وتكون النتيجة عبارة عن مجموعة تضم كل عناصر المجموعتين، ويرمز لها رياضيا هكذا:**

R1 Ć R2

حتى نستوعب المفهوم أكثر سنورد الجدولين التاليين بنفس البنية:



R1 : الفوج الأول من العمال

Code	Name	Age	Address
P1E1	Ahmed	26	Kuwait
P1E2	Idriss	24	Morocco
P1E3	Kamal	25	Egypt

R2 : الفوج الثاني من العمال

Code	Name	Age	Address
P2E1	Khalid	24	Tunisia
P2E2	Ismaîl	27	Syria

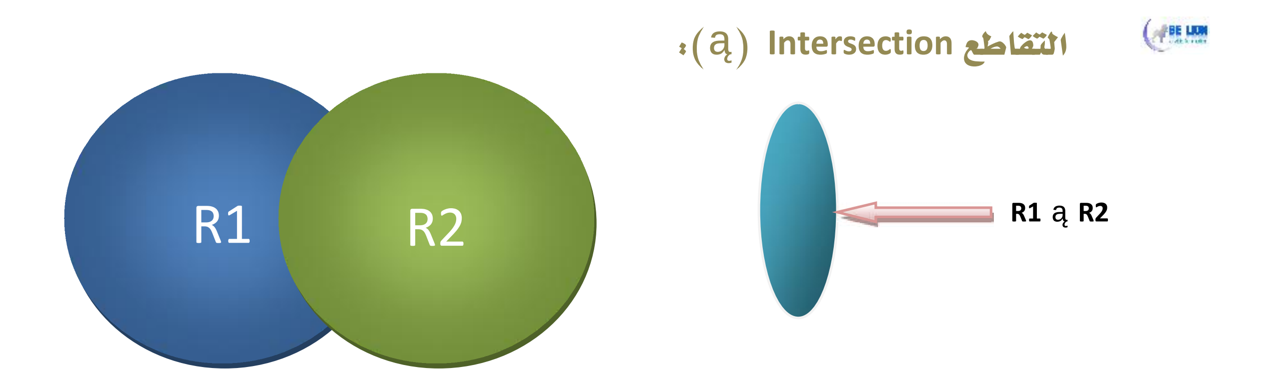
R1ĆR2: اتحاد الوحدتين

Code	Name	Age	Address
P1E1	Ahmed	26	Kuwait
P1E2	Idriss	24	Morocco
P1E3	Kamal	25	Egypt
P2E1	Khalid	24	Tunisia
P2E2	Ismaîl	27	Syria

قائمة أفواج العمال



التقاطع Intersection (ą):



التقاطع هو ناتج ربط جدولين لهما نفس عدد الحقول، ونفس البنية، ويرمز له رياضيا بهذا الرمز R2 ą R1، وهو يضم العناصر المشتركة بين جدولين.

حتى نستوعب المفهوم أكثر سنورد الجدولين التاليين بنفس البنية:

R1: بعض كتب مكتبة "الكندي":

Code	Book
B1	Kalila wa dimna
B2	Moqadimat Ibn Khaldoun
B3	Truth of life
B4	C# Programming
B5	Java Programming

R2: بعض كتب مكتبة "الرضوان":

Code	Book
B4	C# Programming
B13	Health & Body
B3	Truth of life
B15	PHP for beginners
B5	Java Programming

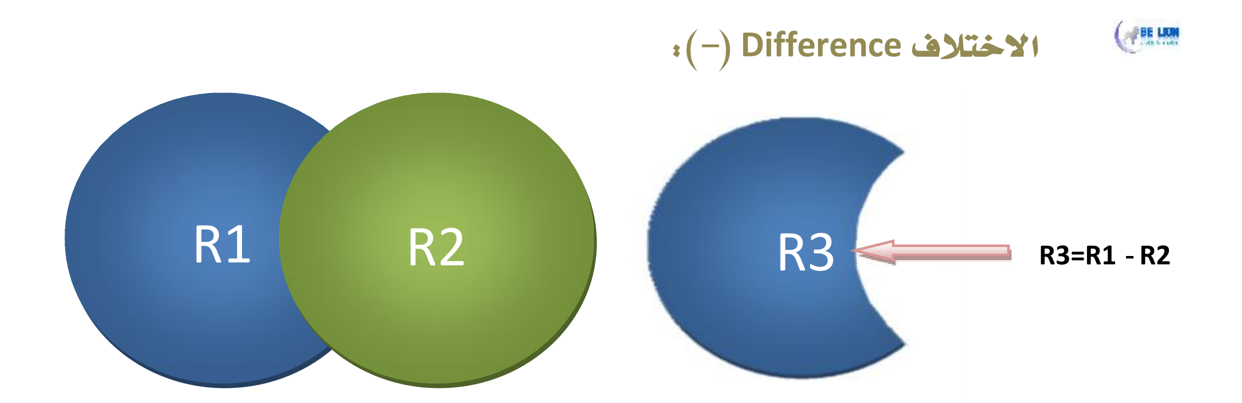


كما تلاحظ لدينا مجموعتان لهما نفس البنية، ولديهما بعض العناصر المشتركة بينهما كما يعرض الجدول التالي:

R1 ą R2: العناصر المشتركة بين المجموعتين:

Code	Book
B4	C# Programming
B3	Truth of life
B5	Java Programming

الاختلاف Difference (−):



الاختلاف هو الفارق الناتج عن طرح مجموعة من مجموعة أخرى، ويشترط أن يكون للمجموعتين (الجدولين) نفس البنية ونفس الحقول.

لنأخذ نفس المثال الأول ونطبق عليه عملية الاختلاف.



R1 : الفوج الأول من العمال

Code	Name	Age	Address
P1E1	Ahmed	26	Kuwait
P1E2	Idriss	24	Morocco
P2E1	Khalid	24	Tunisia
P2E2	Ismaîl	27	Syria
P1E3	Youssef	32	Algeria

R2 : الفوج الثاني من العمال

Code	Name	Age	Address
P2E1	Khalid	24	Tunisia
P2E2	Ismaîl	27	Syria
P2E3	Karim	29	Qatar
P2E4	Mahmoud	31	Arabic Saudia
P2E5	Ibrahim	36	Libya

R1-R2: الفوج الأول ناقص الفوج الثاني:

Code	Name	Age	Address
P1E1	Ahmed	26	Kuwait
P1E2	Idriss	24	Morocco
P1E3	Youssef	32	Algeria

R1-R2: **تعني جلب العناصر الموجودة في** R1 **وغير الموجودة في** R2.



بالمقابل يمكننا القيام بعملية الاختلاف بشكل عكسي، على شكل R2-R1، في هذه الحالة ستكون النتيجة عبارة عن مجموعة تضم فقط العناصر الموجودة في R2 والتي لا توجد في R1 أي كما يعرض الجدول التالي:

R2-R1: الفوج الثاني ناقص الفوج الأول:

Code	Name	Age	Address
P2E3	Karim	29	Qatar
P2E4	Mahmoud	31	Arabic Saudia
P2E5	Ibrahim	36	Libya

العمليات الأحادية

تطبق هذه العمليات على مجموعة واحدة، وتنقسم إلى:

الانتقاء Selection (σ):

وتعني انتقاء بعض العناصر/الأسطر Rows من مجموعة معينة، مثلا لو عندنا جدول العمال التالي:

ID	Name	Function	City
1	Younes MAADANE	Developper	CasaBlanca
2	Ismaîl WAHBI	Conceptor	CasaBlanca
3	Reda Hamdi	Designer	Rabat
4	Hamid MAKBOUL	Director	CasaBlanca
5	Mohammed ELKHAL	Web MASTER	Agadir



الانتقاء في هذه الحالة يعني الاستعلام عن بعض العمال الموجودين ضمن المجموعة، مثلا:

انتقاء العمال الذين يسكنون في مدينة الدار البيضاء CasaBlanca**:**

ID	Name	Function	City
1	Younes MAADANE	Developper	CasaBlanca
2	Ismaîl WAHBI	Conceptor	CasaBlanca
4	Hamid MAKBOUL	Director	CasaBlanca

انتقاء العمال الذين عندهم صفة مبرمج Developper**:**

ID	Name	Function	City
1	Younes MAADANE	Developper	CasaBlanca



الاسقاط Projection (π):

الفرق بينه وبين الانتقاء هو كون الاسقاط يكون بغرض انتقاء الأعمدة Columns **وليس الأسطر، فمثلا لو عندنا نفس الجدو ل السابق:**

ID	Name	Function	City
1	Younes MAADANE	Developper	CasaBlanca
2	Ismaîl WAHBI	Conceptor	CasaBlanca
3	Reda Hamdi	Designer	Rabat
4	Hamid MAKBOUL	Director	CasaBlanca
5	Mohammed ELKHAL	Web MASTER	Agadir

فإن الاسقاط يعني تحديد بعض الحقول فقط في عملية جلب البيانات، مثلا:



جلب أرقام وأسماء العمال فقط:

ID	Name
1	Younes MAADANE
2	Ismaîl WAHBI
3	Reda Hamdi
4	Hamid MAKBOUL
5	Mohammed ELKHAL

أو جلب مهن العمال فقط:

Function
Developper
Conceptor
Designer
Director
Web MASTER

العمليات الثنائية العلائقية

وهي العمليات التي بإمكاننا القيام بها على مجموعتين أو أكثر.

الجداء الديكارتي (×):

ويكون الناتج عن هذه العملية عبارة عن مجموعة جديدة، تضم خارج جداء كل عنصر من المجموعتين بباقي عناصر المجموعة الأخرى، فمثلا لو عندنا الجدولان التاليان:



R1
1
2
3

R2
10
11

R1 X R2	
1	10
2	10
3	10
1	11
2	11
3	11



القسمة Division (÷):

ويعني قسمة جدول على جدول آخر، بشرط أن تكون حقول الجدول الثاني متواجدة في الجدول الأول، كما يرمز له في الرياضيات ÷، وتزن النتيجة عبارة عن جدول يضم عناصر الجدول الأول التي تضم كل عناصر الجدول الثاني، وصيغتها الرياضية هكذا R3=R1÷R2

نفترض مثلا أن عندي جدولان، الأول يضم قائمة للممثلين والأفلام التي شاركوا فيها، والثاني يضم قائمة للأفلام السينمائية:



R1

Actor_Name	Film
Franck Richard	Sun & Moon
José Melany	Like a boss
Franck Richard	Like a boss
Michel Ravaud	Sun & Moon
Katherine elise	Like a boss
José Melany	Sun & Moon

R2

Film
Sun & Moon
Like a boss

كيف نستطيع جلب أسماء الممثلين الذين شاركوا في كل الأفلام؟

للجواب على هذا السؤال سنقوم بقسمة الجدول الأول R1 على الجدول الثاني R2 ونتيجة القسمة ستضم فقط الممثلين الذين شاركوا في كل الأفلام، أي هكذا:

R1÷R2

Actor_Name
Franck Richard
José Melany



الربط Join (⋈):

وهو من أبرز المفاهيم التي سنراها إن شاء الله مع لغة SQL ويقتضي هذا النوع من العمليات جدولين لهما حقل مشترك من نفس النوع، ويستعمل الربط بغرض البحث عن العناصر الموجودة في الجدولين من خلال تحقق شرط وجود الحقل المشترك بنفس القيمة في الجدولين، ويرمز له رياضيا بالرمز التالي ⋈

لنتأمل المثال الآتي:

R1

ID_Country	Country
1	Egypt
2	Morocco
3	Algeria

R2

ID_Citizen	Citizen	ID_Country
C1	Hamdi	1
C2	Khalid	3
C3	Saîd	2

يمكننا من خلال عملية الربط أن نجلب المواطنين والدول مادام رقم الدولة في المجموعة الأولى يتوافق مع رقم الدولة في المجموعة الثانية، هكذا:

```
SELECT Country, Citizen
FROM R1, R2
WHERE R1.Country_id=R2.Country_id
```



نقوم بجلب اسم المواطن من جدول المواطنين، ونجلب اسم الوطن من جدول الأوطان، بشرط أن تتوافق قيمة الحقل المشترك بين المجموعتين، إن استوعبت هذا فقد مررت لك جزء غير يسير من SQL وأنت لا تشعر

خلاصة الجبر العلائقي:

مما سبق نستنتج بأن الجبر العلائقي هو وسيلة لمخاطبة الجداول / المجموعات، بغية جلب بعض البيانات بطرق متعددة، ويمكننا اختصار ما مضى في الخطاطة التالية:



في الفصل الرابع إن شاء الله سوف نرى كيف نحول هاته المفاهيم الرياضية إلى تقنيات عملية لجلب البيانات.



الفصل الثالث

مدخل إلى لغة SQL



مدخل إلى لغة SQL:

تعريف:

كلمة SQL **هي اختصار ل** Structured Query Language، **وتعني لغة الاستعلامات المرتبة، وتستعمل من أجل إجراء عمليات على قواعد البيانات.**

حتى نستوعب هذا المعنى بصفة دقيقة، فلغة SQL **هي التعبير البرمجي للجبر العلائقي الذي رأيناه في مستهل الكتاب، ظهرت هذه اللغة سنة** 1974**، ثم بعد ذلك في سنة** 1986 **تم اعتمادها من طرف** (ANSI)**، وفي سنة** 1987 **تم اعتمادها من قبل** ISO **لتصبح بذلك اللغة الأكثر شيوعا في أنظمة إدارة قواعد البيانات العلائقية** RDBMS.

بالنسبة لبرنامج Microsoft SQL Server **فهو يستعمل نسخة متطورة من** SQL**، تسمى** Transact SQL **وتكتب اختصارا** T-SQL **وهي تضم المهام التالية:**

مهام لغة T-SQL:

- **لغة لتعريف البيانات** DDL**: وهي اختصار ل** **D**ata **D**efinition **L**anguage، **أي أنها تتيح لنا إنشاء وتعديل وحذف الكائنات (قواعد بيانات** Databases، **جداول** Tables، **المشاهد** Views، **الفهارس** Indexes، **إجراءات مخزنة** Stored Procedures، **قوادح** Triggers،...) **كيفما نشاء في قاعدة بيانات علائقية.**



لغة لمعالجة البيانات DML: وهي اختصار ل Data Manipulation Language، أي أنها تمكننا من انتقاء Select وإضافة Add وتحديث Update وحذف Delete البيانات من قاعدة بيانات علائقية.

لغة للتحكم في البيانات DCL: وهي اختصار ل Data Control Language، أي أنها تمكننا من التحكم في مستخدمي قاعدة البيانات عبر تحديد الصلاحيات.

لغة T-SQL ليست لغة للاستعلام فحسب، بل أيضا لغة برمجية كما سيأتي معنا في الفصل الخامس إن شاء الله.



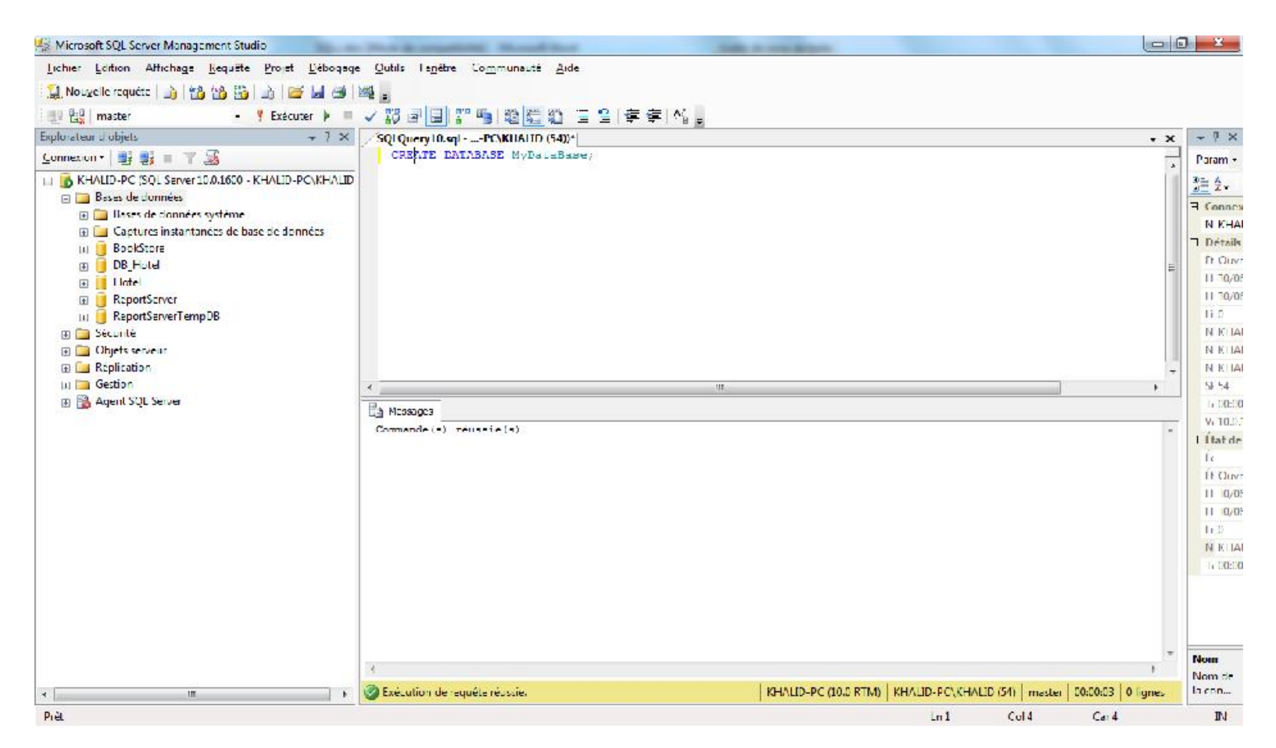
قواعد البيانات

إنشاء قواعد البيانات:

لإنشاء قاعدة بيانات بلغة SQL، فالصيغة كما يلي:

```
CREATE DATABASE MyDatabase ;
```

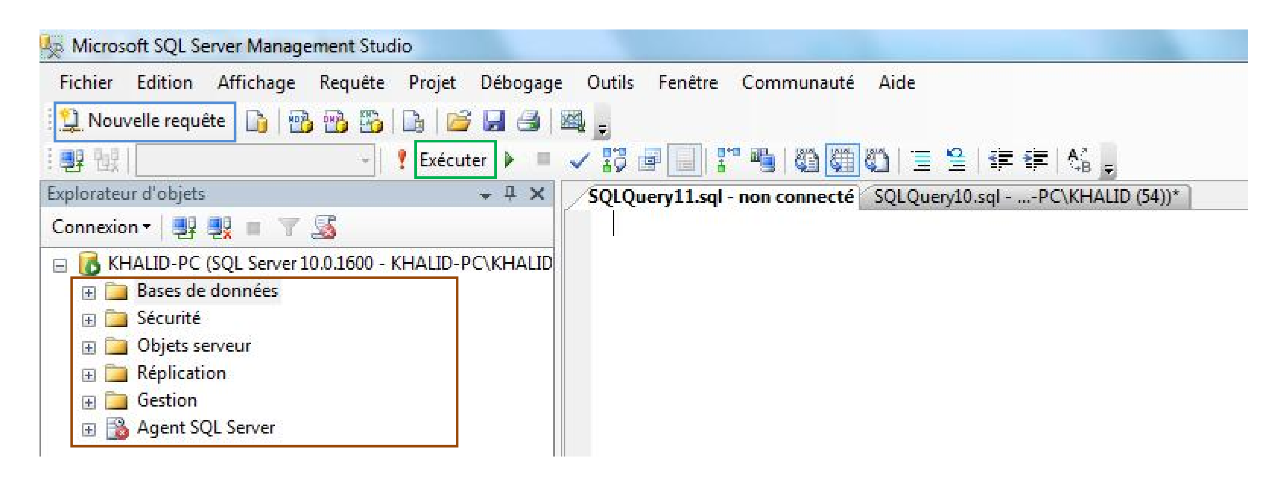
بحيث MyDatabase هو اسم قاعدة البيانات التي نريد إنشاءها.



صورة لواجهة برنامج Microsoft SQL Server



والآن سنعرض أهم القوائم في هذه الواجهة، وسأترجم كل كلمة في البرنامج إلى الانجليزي، لأن أغلب الإخوة يتوفرون على نسخة MS SQL Server بالإنجليزي.



المنطقة المحاطة بالأزرق والتي في النسخة الانجليزية اسمها New Query، تمكننا من فتح صفحة جديدة لكتابة أوامر T-SQL.

المنطقة المحاطة بالأخضر اسمها Execute، وهي خاصة بتنفيذ أوامر SQL المكتوبة في المحرر، بإمكانك الضغط عليها أو على الزر F5.

المنطقة المحاطة باللون البني تضم مستعرض الكائنات Object Explorer، الذي يمكننا من إنشاء الكائنات على مستوى الخادم Server، قواعد بيانات، جداول، ... بالإضافة إلى عمليات الحماية Security وإدارة المستخدمين Users والإتصالات Conenctions.

بإمكاننا أيضا إنشاء قاعدة البيانات يدويا عبر المعالج، من غير حاجة إلى كتابة الأمر CREATE DATABASE.



حذف قاعدة البيانات:

لحذف قاعدة بيانات بواسطة أوامر SQL، نكتب:

```
DROP  DATABASE   MyDatabase ;
```

الجداول Tables

الجداول عبارة عن وحدات لتخزين البيانات على شكل مصفوفة ثنائية الأبعاد (تتكون من أسطر Rows وأعمدة Columns).

قواعد حول التسمية:

عند إنشاء جدول، يلزم عموما احترام مايلي:

- أن لا يكون اسم الجدول كلمة محجوزة في لغة SQL.
- أن لا يبتدئ برقم أو برمز ماعدا بعض الرموز لن نوردها لأنها ليست موحدة بين البرامج، فمثلا برنامج ORACLE يسمح بأن يبدأ اسم الجدول بالرمز $، في حين لا يسمح برنامج SQL Server بذلك.



إنشاء الجداول:

لإنشاء جدول بواسطة أوامر SQL فالصيغة كما يلي:

```
CREATE TABLE MyTable (
                        ID INT,
                        FullName VARCHAR(50),
                        BirthDate DATETIME)
```

الأمر أعلاه يقوم بإنشاء جدول اسمه MyTable ويتكون من حقول ثلاثة، الأول نوعه رقمي، الثاني نوعه نصي يتسع ل 50 حرف، والأخير من نوع التاريخ DataTime.

حذف الجداول:

لحذف جدول نقوم بكتابة الأمر التالي:

```
DROP TABLE MyTable;
```

بحيث MyTable هو اسم الجدول المراد حذفه.

تعديل الجداول:

لإضافة بعض الحقول إلى جدول ما فالصيغة دائما هكذا:

```
ALTER TABLE MyTable ADD Age int;
```

هذا إذا أردنا إضافة حقل واحد للجدول عن طريق أوامر SQL، فقط نكتب بعد الكلمة ADD اسم الحقل ونوعه لتتم إضافته إلى الجدول بعد تنفيذ الأمر.



أما إذا أردنا إضافة مجموعة من الحقول دفعة واحدة، نفصل بينها بفاصلة هكذا:

```
ALTER TABLE MyTable ADD Age int, Address VARCHAR(250);
```

أنواع البيانات:

كل حقل من حقول أي جدول له بالضرورة نوع معين من البيانات، حسب القيمة المراد تخزينها فيه، وتنقسم أنواع البيانات إجمالا إلى:

الأنواع الرقمية:

وتستعمل لتخزين القيم الرقمية، مثلا لو عندنا حقل العمر Age في إحدى الجداول، فحتما علينا اختيار نوع رقمي لتخزين قيم الأعمار، والأنواع الرقمية بدورها تنقسم إلى:

- BIGINT: **ويستعمل لتخزين القيم الرقمية التي مجالها محصور بين** -2^{63} **وبين** $2^{63}-1$ ، **وهذا النوع من البيانات يستعمل 8 بايتات** (8 Bytes) **للتخزين البيانات.**
- INT: **يستعمل أربع بايتات** (4 Bytes) **لتخزين البيانات ومجاله الرياضي يبتدئ من** -2^{31} **وينتهي إلى** $2^{31}-1$
- SMALLINT: **يستعمل 2 بايت** (2 Bytes) **لتخزين البيانات ومجاله الرياضي يبتدئ من** -2^{15} **وينتهي إلى** $2^{15}-1$
- TINYINT: **يستعمل بايت واحد لتخزين البيانات، ومجاله الرياضي محصور بين 0 و** 255.



- REAL: يستعمل لحفظ البيانات من نوع أرقام عشرية، وهو يحتاج إلى 4 بايت للتخزين.
- FLOAT: يستعمل لحفظ البيانات من نوع أرقام عشرية، وهو يحتاج إلى 8 بايت للتخزين.
- DECILMAL: لتخزين القيم العشرية التي يمتد مجالها من 999،99999 - إلى 999،99999
- XML: لتخزين وثائق من نوع الإكس أم أل (XML Documents).

الأنواع النصية:

تستخدم لحفظ البيانات من نوع نصي، على سبيل المثال لو عندنا حقل لحفظ اسم أو عنوان أو أي قيمة نصية، فيلزم أن نختار نوع بيانات نصي، وتنقسم الأنواع النصية إلى:

- VARCHAR(n): وهو من أهم الأنواع النصية، ويسمح بتخزين 8000 بايت من البيانات، وبإمكانك تحديد عدد الأحرف الممكن تخزينها عن طريق تغيير n بالقيمة الرقمية المراد إعطاؤها.
- CHAR(n): يسمح بتخزين النصوص حسب القيمة الرقمية المكتوبة بين القوسين، القيمة الافتراضية لهذا النوع 1، وقيمته القصوى 8000.
- NCHAR(n): لتخزين النصوص، قيمته القصوى 4000.



التاريخ والوقت:

نحتاج هذا النوع من البيانات لحفظ بعض القيم التي تكون على شكل تاريخ ووقت مثل تاريخ البيع أو الشراء...، ومن أهم أقسامه:

- DateTime: يستخدم هذا النوع من البيانات 8 بايتات.

بعض الأنواع الأخرى:

- Binary: لتخزين البيانات الثنائية Binary Data.
- Image: ويستعمل لتخزين البيانات من نوع بايت، وأيضا لتخزين الصور.
- Bit: يسمح فقط بتخزين القيمتين 0 و 1، وهو يستعمل غالبا حينما نتعامل مع بيانات منطقية .

خاصيات الإدخال Constraints :

وهي مجموعة من الأوامر التي نطبقها على الحقول، من أجل التحقق من القيمة المراد حفظها، وأول هذه الكلمات هي:

- **NOT NULL:**

وتستعمل هذه الخاصية لمنع ترك قيمة حقل معين فارغة، وصيغتها هكذا:

```
CREATE TABLE MyTable (
                        ID INT NOT NULL,
                        FullName VARCHAR(60))
```



ويمكننا تطبيق الخاصية NOT NULL حتى بعد إنشاء الجدول كما يلي:

```
--Duting creation of table
CREATE TABLE MyTanble (
                      ID INT NOT NULL,
                      FullName VARCHAR(60))

--After creation of table :

ALTER TABLE MyTable
ALTER COLUMN ID INT NOT NULL
```

كما تلاحظ قمنا بتحديد اسم ونوع الحقل المراد تطبيق الخاصية عليه.

- **IDENTITY:**

هذا النوع من الكلمات يطبق فقط على الحقول التي يكون نوعها رقميا من أجل جعل قيمها تزداد تلقائيا عند إضافة أي سطر جديد، مثلا لو أنشأت الجدول التالي وطبقت هذه الكلمة على الحقل ID، فسوف تزداد قيمته تصاعديا:

```
CREATE TABLE MyTable (
                        ID INT IDENTITY(1,1),
                        FullName VARCHAR(60))
```

عند إضافة أي سطر جديد، ستلاحظ بأن خانة الحقل ID تمنعك من الكتابة فيها، ولكن بمجرد ما تتجاوزها تقوم برفع قيمة الحقل بواحد تلقائيا، كما تبين الصورة التالية:

	ID	FullName
	1	Khalid
	2	Ahmed
	3	Youssef
▸*	NULL	NULL



تستطيع تغيير القيمة التي يبدأ منها الحقل، وأيضا معامل زيادة القيمة هكذا:

```
CREATE TABLE MyTable (
                        ID INT IDENTITY(2,3),
                        FullName VARCHAR(60))
```

في هذا المثال، ستبدأ قيمة الحقل من الرقم 2، ويكون معامل الزيادة هو 3، أي أن القيمة الأولى ستكون 2، والقيمة الثانية ستكون 5، والثالثة ستكون 8 ...إلخ.

- **PRIMARY KEY:**

تطبق هذه الخاصية على الحقل لجعل قيمته متفردة، غير قابلة للتكرار ولا ينبغي أن تترك قيمة الحقل الذي تطبق عليه هذه الخاصية فارغة أي ينبغي أن تكون NOT NULL، فمثلا لو عندنا جدول يضم بيانات المواطنين، فمن غير شك هنالك حقل يميز كل مواطن عن غيره، في هذه الحالة الحقل الذي سيكون مفتاحا أساسيا PRIMARY KEY هو الحقل الذي يضم أرقام بطاقات التعريف الوطنية، لأنه من الممكن أن تجد أكثر من مواطن يحملون نفس الاسم، ولكن يستحيل أن يحملوا نفس رقم بطاقة التعريف، عند التعامل مع جداول متصلة فيما بينها عن طريق العلاقات يصبح إلزاميا استخدام المفاتيح الأساسية.

لجعل أحد الحقول مفتاحا أساسيا، نكتب الصيغة التالية:

```
CREATE TABLE MyTable (
                        ID INT PRIMARY KEY,
                        FullName VARCHAR(60))
```



ونستطيع تسمية خاصية التحقق PRIMARY KEY أثناء تطبيقها على إحدى الحقول بالصيغة التالية:

```
CREATE TABLE MyTable (
                      ID INT CONSTRAINT PK_MyConstraint PRIMARY KEY,
                      FullName VARCHAR(60))
```

في هذه الحالة أسمينا الخاصية PRIMARY KEY المطبقة على الحقل ID ب PK_MyConstraint.

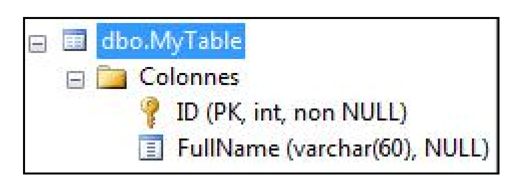
نستطيع أيضا تطبيق هذه الخاصية حتى بعد إنشاء الجدول، عن طريق الخاصية ALTER TABLE، أي هكذا :

```
--creation of table
CREATE TABLE MyTable (
                      ID INT NOT NULL,
                      FullName VARCHAR(60))

--Add primary key constraint :

ALTER TABLE MyTable ADD CONSTRAINT PK_MyConstraint PRIMARY KEY(ID)
```

لو ذهبت إلى الجدول في القائمة، ودخلت إلى الأعمدة Columns ستجده بالشكل التالي:



رمز المفتاح للدلالة على أن الحقل ID حقل أساسي PRIMARY KEY.



UNIQUE:

تستعمل هذه الخاصية لمنع قيمة حقل معين من التكرار، وبالتالي لا يمكن أن يضم نفس الحقل قيمة أكثر من مرة، الفرق بين الخاصية UNIQUE وبين الخاصية PRIMARY KEY، أن هذه الأخيرة لا يمكن تطبيقها على الحقول التي تسمح بالقيم الفارغة NULL، بينما تسمح الخاصية UNIQUE بالقيمة NULL، ويمكننا تطبيق هذه الخاصية على الحقل هكذا:

```
CREATE TABLE MyTable (
                        ID INT UNIQUE,
                        FullName VARCHAR(60))
```

ويمكننا تطبيقها على جدول منشأ مسبقا كما يلي :

```
--creation of table
CREATE TABLE MyTable (
                      ID INT UNIQUE,
                      FullName VARCHAR(60))

--Add unique constraint :

ALTER TABLE MyTable ADD CONSTRAINT U_MyConstraint UNIQUE (ID)
```

REFERENCE:

هذه الخاصية تطبق على الحقول الأجنبية القادمة من جدول آخر، وتستعمل لتعريف الحقل الأجنبي وتحديد الجدول القادم منه.



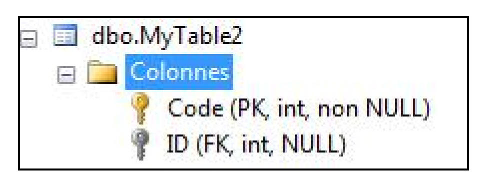
في هذا المثال سننشئ جدولين، ونحاول تطبيق الخاصية REFERENCE على الجدول الثاني:

```
--creation of the first table
CREATE TABLE MyTable1 (
                        ID INT NOT NULL PRIMARY KEY ,
                        FullName VARCHAR(60))
```

```
--creation of the second table
CREATE TABLE MyTable2 (
                  Code INT NOT NULL PRIMARY KEY ,
                  ID INT CONSTRAINT PK_MyTable2  REFERENCES MyTable1(ID)
                        )
```

الجدول الثاني يتكون من حقلين، الأول اسمه Code وهو الحقل الأساسي، والثاني اسمه ID وهو أجنبي قادم من الجدول الأول.

لو ذهبت إلى الجدول في القائمة، ودخلت إلى الأعمدة Columns ستجده بالشكل التالي:



المفتاح الذهبي للدلالة على أن الحقل أساسي PRIMARY KEY، والمفتاح الرمادي للدلالة على أن هذا الحقل أجنبي FOREIGN KEY.

ونستطيع أيضا تطبيق الخاصية REFERENCES على جدول تم إنشاؤه مقدما، وتكون الصيغة هكذا:




```
ALTER TABLE MyTable2
ADD CONSTRAINT FK_Constraint
FOREIGN KEY (ID)
REFERENCES MyTable1(ID)
```

بحيث FK_Constraint هو اسم الخاصية REFERENCES التي نحن بصدد إنشائها، أما ID الأولى المقرونة ب FOREIGN KEY فهو اسم الحقل الأجنبي في الجدول الثاني، و ID الثانية المقرونة ب REFERENCES هو اسم الحقل الأجنبي في الجدول المصدر (في حالتنا هذه هو الجدول الأول).

لتفادي وقوع مشاكل، يستحسن إضافة الخاصية REFERENCES بعد إنشاء الجداول، لأن الجداول التي تضم الحقول الأساسية يجب أن تنشأ أولا قبل الجداول التي تضم الحقول الأجنبية.

- **DEFAULT :**

الغاية من هذه الخاصية هي تحديد قيمة افتراضية للحقل المطبقة عليه تفاديا للفراغ NULL، وطبعا هذه القيمة الافتراضية لا تحفظ إلا إذا ترك المستخدم قيمة الحقل فارغة، ولتطبيق هذه الخاصية على حقل معين فالصيغة كما يلي:

```
CREATE TABLE MyTable(MyColumn nvarchar(25) DEFAULT 'UnKnown')
```

إذا كان نوع الحقل رقميا، نكتب القيمة من غير علامات التنصيص، إذا أردنا تطبيق هذه الخاصية على حقل موجود مسبقا فالصيغة كما يلي:



```
ALTER TABLE MyTable ADD CONSTRAINT D_Constraint DEFAULT 'UnKnown' For
MyColumn
```

- **CHECK:**

تستعمل هذه الخاصية للتحقق من قيمة الحقل قبل إدخاله، فإن كانت القيمة تتوافق مع الشرط المصحوب بالخاصية تمت عملية الإدخال، وإن كانت القيمة تخالف الشرط يتم منع عملية الإدخال، وصيغتها هكذا:

```
CREATE TABLE Person(
       CIN CHAR(9),
       FullName NVARCHAR(75),
       AGE INT CONSTRAINT C_Constraint CHECK (AGE BETWEEN 5 AND 160))
```

في هذا المثال سيسمح فقط بإدخال قيمة العمر المحصورة بين 5 سنوات و 160 سنة، ويمكنك تعويض الشرط الموجود بين القوسين بأي شرط تريد.

يمكننا إضافة الخاصية CHECK حتى بعد إنشاء الجدول، بالصيغة التالية:

```
ALTER TABLE Person

ADD CONSTRAINT C_Constraint

CHECK (Age BETWEEN 5 AND 160)
```

حذف خاصيات الإدخال:

لحذف خاصيات الإدخال، نكتب الصيغة التالية:




```
ALTER TABLE Person
DROP CONSTRAINT C_Constraint
```

بحيث Person هو اسم الجدول، و C_Constraint اسم خاصية الإدخال المراد حذفها.

المشاهد Views:

المشاهد Views عبارة عن جداول وهمية، والتي لها نفس دور الجداول الحقيقية إلا أنها أخف وأسرع لأنها لا تحتوي على البيانات وإنما تحتوي فقط على الاستعلام الذي يجلب البيانات من الجداول ، بالإضافة إلى هذه المزية تمكننا المشاهد من تحديد عدد الحقول المراد تضمينها في الاستعلام، ناهيك عن كونها تمكننا من جلب البيانات من مجموعة من الجداول المترابطة وتجميعها على شكل جدول واحد.

إنشاء المشاهد:

لإنشاء مشهد View معين، فالصيغة العامة كما يلي:

```
CREATE VIEW View_Name
AS Query
```

بحيث View_Name هو اسم المشهد المراد إنشاؤه، وهذا مثال لكيفية إنشاء مشهد يجلب بيانات الموظفين الذين يسكنون مدينة الرياض:

```
CREATE VIEW Emp_Riad
AS
SELECT ID, FullName, Age FROM Employee WHERE [City]='Riad'
```



لو أردت رؤية المشهد الذي قمت بإنشائه اذهب إلى قاعدة البيانات، ستجد في التبويب الذي يوجد فيه الجداول، تبويبا اسمه Views، ادخل إليه وستجد المشهد بالإسم الذي أعطيته إياه.

- DB_Hotel
  - Schémas de base de données
  - Tables
  - Vues
    - Vues du système
    - dbo.Emp_Riad
  - Synonymes
  - Programmabilité
  - Service Broker
  - Stockage
  - Sécurité

ادخل إليها ستجدها شبيهة جدا بالجداول Tables.

بإمكانك إنشاء المشاهد أيضا من هناك، من دون الحاجة إلى كتابة أوامر SQL، فقط اتبع مراحل الإنشاء، بعد الضغط بيمين الفأرة على Views واختيار New View.

حذف المشاهد Views:

لحذف مشهد معين، قم بالضغط عليه بيمين الفأرة واختر الأمر Delete، أو اكتب الأمر التالي:

```
DROP VIEW View_Name
```



الفهارس Indexes:

هي شبيهة بالفهارس التي تعرفها، والتي تكون موجودة غالبا في أواخر صفحات الكتاب، كما تعلم فهذه الفهارس توضع من أجل تمكين القارئ من الوصول إلى الموضوع الذي يبحث عنه بكل سرعة.

نفس الشيء ينطبق على الفهارس في لغة SQL**، إذ أن الغاية من إنشاء الفهارس هو تسريع عمليات جرد البيانات** (Select)**، لكن عيبها هو أنها تبطئ عمليات تحديث البيانات من إضافة وحذف وتعديل، لأن كل عملية تحديث تتسبب في إعادة إنشاء الفهرس، لهذا وحسن التقليل منها.**

إنشاء الفهارس:

لإنشاء فهرس جديد على إحدى الحقول في جدول ما، فإن الصيغة كما يلي:

```
CREATE INDEX Index_Name
ON Table_Name (Column_Name Desc | Asc)
```

بحيث Index_Name **هو اسم الفهرس، و** Table_Name **اسم الجدول المعني بالأمر، و** Column_Name **اسم العمود / الحقل المراد عمل فهرسة له.**

أما Desc **و** Asc **فهي تحدد طريقة جرد البيانات وهي إما تصاعدية** Ascending **أو تنازلية** Descending.



وهذا مثال لإنشاء فهرس:

```
CREATE INDEX MyIndex
ON MyTable (ID Desc)
```

حذف الفهارس:

لحذف فهرس بواسطة أوامر SQL، فالصيغة كما يلي:

```
DROP INDEX MyIndex
ON MyTable
```

معالجة البيانات:

تحدثنا عن مفهوم معالجة البيانات في مستهل الفصل الثاني، وقلنا بأنها تشمل كل عمليات الإضافة والتعديل والحذف التي تطال الجداول.

إضافة البيانات Insert إلى جدول:

مثلا عندي جدول يضم قائمة الموظفين، وأريد أن أضيف موظفا جديدا إلى هذا الجدول، سأفترض أن الجدول يضم حقلين فقط وهما: رقم الموظف واسمه الكامل، لفعل ذلك فالصيغة كما يلي:

```
INSERT INTO MyTable
(ID, FullName)
VALUES
(1, 'Khalid')
```



بحيث MyTable هو اسم الجدول الذي نريد إضافة البيانات إليه، و ID و FullName هما الحقلان المشكلان لهذا الجدول، أَيُّ حقل من حقول الجدول لا يكتب بين الأقواس توضع فيه القيمة NULL افتراضيا.

إذا كان أحد الحقول يطبق الخاصية IDENTITY التي تجعل الرقم يزداد تلقائيا، فلا ينبغي أن نضعه مع الحقول.

إذا كنا متأكدين من ترتيب الحقول، فبإمكاننا إلغاء ذكر الحقول في الصيغة والاكتفاء فقط بإدخال القيمة، كما نعرض هنا:

```
INSERT INTO MyTable
VALUES
(1, 'Khalid')
```

نسخ البيانات من جدول إلى آخر

بإمكاننا نسخ بيانات جدول معين، ونقلها إلى جدول ثان والصيغة كما يلي:

```
INSERT INTO MyTable2 (ID,FullName)
SELECT ID,Fullname
FROM MyTable1
```

في السطر الأول قمنا بتحديد اسم الجدول المراد نسخ البيانات إليه مع تحديد أسماء الحقول، ثم بعد ذلك نقوم بجلب قيم الحقول من الجدول المصدر عن طريق الكلمة Select التي سنراها بالتفصيل فيما سيأتي إن شاء الله.



إضافة البيانات إلى جدول في نفس لحظة إنشائه:

بإمكاننا تعبئة جدول في لحظة إنشائه بعناصر جدول آخر موجود مسبقا، للقيام بذلك فالصيغة كما يلي:

```
SELECT *
INTO NewTable
FROM OldTable
```

النجمة * تعني جلب جميع الحقول، إذا قمت بتنفيذ هذا الكود سوف تكون النتيجة عبارة عن جدول جديد، اسمه NewTable معبأ بنفس بيانات الجدول OldTable.

حذف البيانات Delete:

لحذف جميع البيانات من جدول معين، فالصيغة كما يلي:

```
DELETE FROM MyTable;
```

إذا أردنا حذف بعض العناصر فقط فيلزمنا إضافة الكلمة الشرطية WHERE، التي تخول لنا تحديد الشرط الذي بتحققه تتم عملية الحذف.

وكمثال لذلك نفترض أن لدينا جدولا يضم مجموعة من الموظفين، ونريد حذف الموظفين الذين يقطنون بمدينة الرباط:

```
DELETE FROM Employee
WHERE Adress= 'Rabat';
```

مثال آخر يقوم بحذف الموظفين الذين عمرهم أكبر من 60 سنة:




```sql
DELETE FROM Employee
WHERE Age > 60;
```

تعديل البيانات Update Data:

في معظم الأمثلة القادمة سنشتغل على جدول الموظفين، لهذا يستحسن أن تنشئه الآن لتجريب الأوامر، بإمكانك إنشاؤه يدويا عن طريق نافذة التصميم، أو عبر أوامر SQL التي رأيناها آنفا.

بالنسبة لحقول الجدول فلست ملزما باتباع نفس الحقول التي أستعملها، لأنني أغير الحقول وفق المفهوم الذي أعرضه، على العموم تستطيع ابتداءً إنشاء جدول الموظفين بالحقول التي أوردها في استعلام الإنشاء التالي:

```sql
CREATE TABLE Employee
(ID INT NOT NULL PRIMARY KEY,
FullName NVARCHAR(40),
Adress NVARCHAR(255),
Age TINYINT)
```

للقيام بعملية تعديل البيانات في جدول ما، فإن الصيغة تكون كما يلي:

```sql
UPDATE Employee
SET FullName='UnKnown', Adress='Jeddah'
```

المثال أعلاه يقوم بتعديل بيانات كل العناصر الموجودة داخل جدول Employees، وبالتالي لجعل التعديل يشمل بعض العناصر دون غيرها يلزمنا تحديد شرط التعديل والذي يأتي بعد الكلمة WHERE، في المثال الآتي سنغير معلومات الموظف الذي يحمل الرقم 1:




```
UPDATE Employee
SET FullName='UnKnown', Adress='Jeddah'
WHERE ID=1
```

وهذا مثال آخر يقوم بتغيير أسماء الموظفين القاطنين بمدينة الرياض، وتعويض أسمائهم ب "حمزة":

```
UPDATE Employee
SET FullName='Hamza'
WHERE Adress='Riad'
```

جرد البيانات Select:

رأينا فيما سلف كيف يقوم الأمر Select بجلب البيانات من الجداول، الآن إن شاء الله سنتعرف عليه أكثر، وكنقطة للبداية سنورد صيغة هذا الأمر:

```
SELECT Field1, Field2, ...
FROM TableName
WHERE Condition
```

في الجزء الأول من Select نقوم بتحديد الحقول التي نريد استعراض قيمها، وفي الجزء الثاني نحدد الجدول المراد جلب البيانات منه، وفي الجزء الثالث نستطيع أن نضع شرطا يضبط جلب البيانات، كما يظهر المثال التالي:

```
SELECT ID, FullName, Adress
FROM Employee
WHERE Adress='Mekkah'
```



ستكون النتيجة كما يلي:

	ID	FullName	Adress
1	2	Hamid MAKBOUL	Mekkah
2	3	Mohamed ALQAHTANI	Mekkah
3	4	Youssef Ahmed	Mekkah

لجلب جميع الحقول، نستغني عن كتابة الأسماء ونعوضها بالرمز نجمة (*) الذي يعني كل الحقول:

```
SELECT *
FROM Employee
WHERE Adress='Mekkah'
```

الاستعلام السابق سيعطي نفس النتيجة.

بمقدورنا وضع أكثر من شرط بعد الكلمة WHERE، كما يوضح المثال التالي:

```
SELECT *
FROM Employee
WHERE Adress='Mekkah'
And Age < 30
```

هذه المرة سيجلب الموظفين القاطنين في مكة المكرمة والدين عمرهم أصغر من 30 سنة.

وهذا مثال آخر يقوم بجلب الموظفين القاطنين في مكة المكرمة أو المدينة المنورة:

```
SELECT *
FROM Employee
WHERE Adress='Mekkah'
OR Adress='Madina'
```



سيقوم الاستعلام أعلاه بجلب كل الموظفين القاطنين بمكة والمدينة، والنتيجة كما يلي:

	ID	FullName	Adress	Age
1	1	Karim Hamdi	Madina	24
2	2	Hamid MAKBOUL	Mekkah	23
3	3	Mohamed ALQAHTANI	Mekkah	35
4	4	Youssef Ahmed	Mekkah	42
5	5	Nihad Chawqi	Madina	21

بإمكاننا تغيير أسماء الحقول عند العرض، ولفعل ذلك نحدد اسم الحقل المراد عرضه بعد الكلمة AS:

```
SELECT ID AS الرقم,
FullName AS  [الاسم الكامل],
Adress AS العنوان,
Age AS  العمر
FROM Employee
```

والنتيجة كما يلي:

	الرقم	الاسم الكامل	العنوان	العمر
1	1	Karim Hamdi	Madina	24
2	2	Hamid MAKBOUL	Mekkah	23
3	3	Mohamed ALQAHTANI	Mekkah	35
4	4	Youssef Ahmed	Mekkah	42
5	5	Nihad Chawqi	Madina	21



دمج الحقول Concatenation:

أحيانا نريد إظهار قيم بعض الحقول مرتبطة فيما بينها كما لو أنها قيمة واحدة، تسمى هذه العميلة بالدمج Concatenation، للقيام بذلك نفرق بين الحقول بعلامة +، كما يعرض المثال التالي:

```
SELECT  FullName + '   Is From:   ' + Adress as 'About this emplyee'
From Employee
```

ستكون النتيجة هكذا :

	About this emplyee
1	Karim Hamdi Is From: Madina
2	Hamid MAKBOUL Is From: Mekkah
3	Mohamed ALQAHTANI Is From: Mekkah
4	Youssef Ahmed Is From: Mekkah
5	Nihad Chawqi Is From: Madina

جرد الأسطر الأولى SELECT TOP:

أحيانا نريد إظهار الأسطر الأولى فقط من جدول ما، للقيام بذلك نستخدم الكلمة TOP بعد الكلمة SELECT ثم نحدد عدد الأسطر الأولى المراد استعراضها، كما يوضح المثال التالي:

```
SELECT TOP 3 *
FROM Employee
```



نتيجة المثال أعلاه كما يلي:

	ID	FullName	Adress	Age
1	1	Karim Hamdi	Madina	24
2	2	Hamid MAKBOUL	Mekkah	23
3	3	Mohamed ALQAHTANI	Mekkah	35

كما تلاحظ قام بجلب الثلاثة الأوائل فقط.

جلب البيانات عشوائيا RANDOM SELECT:

إذا أردنا جلب عناصر عشوائية فعلينا النداء على الدالة NewId() مصحوبة بأمر الترتيب Order By الذي سنراه لاحقا، وكمثال لذلك نورد:

```
SELECT  *
FROM Employee
ORDER BY newid()
```

في كل مرة تقوم بتنفيذ هذا الاستعلام، ستلاحظ بأن البيانات تأتي عشوائيا، قد تحتاج هذا الأمر إذا كنت بصدد إنجاز برنامج يحتاج إلى تولد اختيارات عشوائية مثل مسابقة "من سيربح المليون؟".

جلب البيانات غيرمكررة SELECT DISTINCT:

تمكننا الدالة DISTINCT من جلب البيانات من حقل معين مع تفادي التكرار، بحيث لو كان يضم الحقل نفس القيمة في أكثر من سطر، تجلب هذه القيمة مرة واحدة، وصيغة هذه الدالة كما يلي:




```
SELECT DISTINCT FullName
FROM Employee
```

لنفترض أن جدول الموظفين يضم البيانات التالية:

ID	FullName
1	Khalid
2	Ahmed
3	Khalid
4	Karim

بعد تنفيذ الاستعلام أعلاه، ستكون البيانات المجلوبة كما يلي:

FullName
Khalid
Ahmed
Karim

جلب البيانات المشابهة LIKE:

لعلك تتساءل كيف تقوم محركات البحث (غوغل مثلا) بجلب نتائج مشابهة للكلمة التي تبحث عنها، فيما يلي سنتعرف على كلمة تقوم بنفس العمل، إنها الكلمة Like.

هذا المثال يقوم بجلب الموظفين الذين يبدأ اسمهم بحرف "M":

```
SELECT  *
FROM Employee
WHERE FullName like 'M%'
```



الاستعلام أعلاه يعني جرد الموظفين الذين يبدأ اسمهم بحرف M، ورمز النسبة المئوية % يعني لا يهمنا ما يأتي بعد الحرف M، وعليه فالنتيجة ستكون كما يلي:

	ID	FullName	Adress	Age
1	3	Mohamed ALQAHTANI	Mekkah	35

وهذا مثال آخر يقوم بجلب الموظفين الذين تنتهي أسماؤهم بحرف بالحرف "I":

```
SELECT  *
FROM Employee
WHERE FullName Like '%I'
```

النتيجة كما يلي:

	ID	FullName	Adress	Age
1	1	Karim Hamdi	Madina	24
2	3	Mohamed ALQAHTANI	Mekkah	35
3	5	Nihad Chawqi	Madina	21

وهنا مثال لجلب الموظفين الذين تضم أسماؤهم حرف "N":

```
SELECT  *
FROM Employee
WHERE FullName Like '%N%'
```



ترتيب البيانات ORDER BY:

نريد جرد الموظفين مرتبين حسب أعمارهم، من الأكبر إلى الأصغر، للقيام بذلك نستخدم الكلمة ORDER BY **التي تمكننا من ترتيب البيانات إما تصاعديا أو تنازليا:**

```
SELECT  *
FROM Employee
ORDER BY Age DESC
```

الاستعلام أعلاه يقوم بجرد الموظفين مرتبين وفق أعمارهم تنازليا من الأكبر إلى الأصغر، والنتيجة كما يلي:

	ID	FullName	Adress	Age
1	4	Youssef Ahmed	Mekkah	42
2	3	Mohamed ALQAHTANI	Mekkah	35
3	1	Karim Hamdi	Madina	24
4	2	Hamid MAKBOUL	Mekkah	23
5	5	Nihad Chawqi	Madina	21

ويمكننا القيام نفس العملية تصاعديا، أي من الأصغر إلى الأكبر فقط بتبديل الكلمة DESC (**وهي اختصار للكلمة** descendant) **بالكلمة** ASC (**وهي اختصار للكلمة** ascendant).

وهذا مثال للتوضيح:

```
SELECT  *
FROM Employee
ORDER BY Age ASC
```



ويمكننا القيام بالترتيب المتعدد عن طريق تحديد العديد من الحقول بعد الكلمة ORDER BY، كما يعرض هذا المثال:

```
SELECT  *
FROM Employee
ORDER BY Age, Adress
```

الدوال Functions:

الدوال عبارة عن مجموعة من البرامج المنجزة مسبقا، والتي تتيح للمستخدم بعض الخدمات التي تغنيه عن كتابتها بواسطة الأكواد، من قبيل الدوال الرياضية (المجموع Sum، المتوسط Average، ..)، التي قد يحتاجها المستخدم.

توفر لغة SQL مجموعة كبيرة من الدوال لإنجاز بعض المهام التي يحتاجها المستخدم، في هذا الجزء سنورد بعض الدوال الأكثر شيوعا والتي قد تحتاجها مستقبلا.

الدوال التجميعية Aggregate Functions:

الدالة COUNT:

وتعيد لنا هذه الدالة عدد عناصر جدول معين، وصيغتها هكذا:

```
SELECT COUNT(*)
FROM Employee
```



النتيجة عبارة عن قيمة رقمية تمثل عدد العناصر التي يجلبها الاستعلام.

الدالة SUM

تعيد لنا الدالة SUM، قيمة تمثل مجموع قيم الحقل الرقمي المحدد، فمثلا لو افترضنا أنه لدينا جدول الموظفين التالي:

ID	Name	Job	Hours
1	Mohamed	Developper	56
2	Hamid	Web Master	45
3	Younes	Conceptor	78
4	Khalid	Designer	84

لو أردنا معرفة مجموع الساعات التي اشتغلها هؤلاء الموظفون، فعلينا استعمال الدالة SUM كما يلي:

```
SELECT SUM(Hours)
FROM Employee
```

لابد أن يكون نوع الحقل الذي نطبق عليه الدالة SUM من نوع رقمي.

الدالة AVG

وتقوم هذه الدالة بحساب متوسط الحقل المحدد، وكما هو معلوم في الرياضيات فالمتوسط يساوي مجموع قيم العناصر مقسوما على عدد العناصر.

صيغة الدالة AVG كما يلي:




```
SELECT AVG(Hours)
FROM Employee
```

النتيجة ستكون عن عبارة قيمة رقمية تمثل مجموع الساعات مقسوم على عدد الموظفين، تطبق هذه الدالة فقط على الحقول الرقمية.

بإمكاننا الحصول على نفس النتيجة من غير استخدام الدالة AVG، وذلك كما يلي:

```
SELECT SUM(Hours)/ COUNT(Hours)
FROM Employee
```

لأن المتوسط يساوي المجموع مقسوم على عدد العناصر.

الدالة MIN

وتعيد لنا هذه الدالة أصغر قيمة في الحقل المحدد، وصيغتها كما يلي:

```
SELECT MIN(Hours)
FROM Employee
```

الدالة MAX

وتعيد لنا هذه الدالة أكبر قيمة في الحقل المحدد، وصيغتها كما يلي:

```
SELECT MAX(Hours)
FROM Employee
```



تجميع البيانات GROUP BY:

تستعمل الكلمة GROUP BY مع إحدى الدوال التجميعية Aggregate functions، وتمكننا من تجميع البيانات وفق حقل معين، مثلا تحديد عدد الموظفين القاطنين في كل مدينة، أو عدد التلاميذ الذين يدرسون في كل فصل...إلخ.

وصيغتها كما يلي:

```
SELECT
COUNT(Adress) as 'عدد سكان كل مدينة'
FROM Employee
GROUP BY Adress
```

وهذا مثال توضيحي لاستعمال الكلمة GROUP BY:

```
SELECT Age as 'السن',
COUNT(Age) as 'عدد الموظفين الذين لهم هذا السن'
FROM Employee
GROUP BY Age
```

بعد تنفيذ هذا الاستعلام، سنحصل على حقلين، الأول يعرض كل الأعمار الموجودة في جدول الموظفين، والحقل الثاني يعرض عدد الموظفين الذين يبلغون كل عمر، وعليه فالنتيجة ستكون هكذا:

	السن	عدد الموظفين الذين لهم هذا السن
1	21	2
2	23	1
3	24	3



شرط التجميع HAVING:

تستخدم الكلمة HAVING بعد الكلمة GROUP BY، لتحديد شرط جلب البيانات، مثلا بعد أن تقوم GROUP BY بتجميع الموظفين حسب المدن، تقوم HAVING بجلب المجموعات القاطنة في مدينة الرياض مثلا.

صيغتها كما يلي:

```
SELECT Adress,
COUNT(Adress)
FROM Employee
GROUP BY Adress
HAVING adress='Riad'
```

الاستعلام أعلاه يقوم بجلب المدن، وأمامها عدد الموظفين القاطنين بها، لكن بعد أن كتبنا HAVING سيقوم بجلب سكان مدينة الرياض فقط.

في المثال التالي، سنجلب كل الأعمار، وأمامها عدد الموظفين البالغين لها، ثم نقوم بالفرز بواسطة HAVING ، لنجلب فقط الأعمار التي يفوق عدد بالغيها أربعة أشخاص:

```
SELECT age,
COUNT(Age )
FROM Employee
GROUP BY age
HAVING COUNT(age)>4
```

قم بتجريب المثال وتتعرف على الكلمة HAVING أكثر.



الدوال الحسابية Arithmetic Functions:

الدالة ABS

الدالة ABS **تعيد لنا القيمة المطلقة** Absolute Value **للحقل المحدد، وصيغتها كما يلي:**

```
SELECT ABS(-67)
```

في هذا المثال سنحصل على القيمة المطلقة للرقم 67-**، وهي** 67.

الدالة SQRT

تعيد لنا هذه الدالة قيمة الجذر المربع للرقم المحدد، وصيغتها مثل الدوال السالفة.

يوجد المزيد من الدوال الرياضية، ما أوردناه ليس كل شيء.

الدوال النصية String Functions:

الدالة SUBSTRING

تعيد لنا هذه الدالة جزء من النص أو الحقل المحدد، وصيغتها العامة كما يلي:

```
SUBSTRING(النص أو الحقل المراد اجتزاؤه,بداية الاجتزاء,طول الاجتزاء)
```

المعامل الأول "النص أو الحقل المراد اجتزاؤه" **نعوضه باسم الحقل الذي نريد اقتطاع النص منه.**



المعامل الثاني "بداية الاجتزاء" نعوضه برقم الحرف الذي نريد ابتداء التقطيع انطلاقا منه.

المعامل الثالث "طول الاجتزاء" نعوضه بعدد الأحرف المراد تقطيعها انطلاقا من بداية الاجتزاء.

وهذا المثال يعرض كيف نستخدم الدالة Substring مع حقل نصي:

```
SELECT FullName AS 'الاسم الكامل',
SUBSTRING(FullName, 3,5) AS 'جزء من الاسم'
FROM Employee
```

في هذا المثال سنحصل على نتيجة متكونة من حقلين، أحدهما يعرض الاسم كاملا، والآخر يعرض خمسة أحرف من الاسم انطلاقا من الحرف الثاني، وعليه فإن النتيجة كما يلي:

	الاسم الكامل	جزء من الاسم
1	Karim Hamdi	rim H
2	Hamid MAKBOUL	mid M
3	Mohamed ALQAHTANI	hamed
4	Youssef Ahmed	ussef
5	Nihad Chawqi	had C

الدالة LEFT

تقوم هذه الدالة باقتطاع النص انطلاقا من اليسار وانتهاء بقيمة طول الاجتزاء، وصيغتها كما يلي:

```
LEFT(الحقل أو النص المراد اجتزاؤه, طول الاجتزاء)
```



سنطبقها على نفس المثال السابق:

```
SELECT FullName AS 'الاسم الكامل',
RIGHT(FullName, 5) AS 'جزء من الاسم'
FROM Employee
```

هذه المرة ستكون النتيجة عبارة عن حقلين، أولهما يعرض الاسم كاملا، والثاني يعرض جزء من الاسم طوله خمسة أحرف ابتداء من اليسار:

	الاسم الكامل	جزء من الاسم
1	Karim Hamdi	mdi
2	Hamid MAKBOUL	OUL
3	Mohamed ALQAHTANI	ANI
4	Youssef Ahmed	med
5	Nihad Chawqi	wqi

الدالة RIGHT

تقوم بنفس دور الدالة السابقة، ولكن انطلاقا من اليمين، بالنسبة لصيغتها فهي أيضا هكذا:

```
RIGHT(اجتزاؤه المراد النص أو الحقل, الاجتزاء طول)
```

لن نورد مثالا لأنها شبيهة بالدالة السابقة.



الدالتان LTRIM و RTRIM

تقوم الدالة RTRIM بمسح الفراغات عن يمين النص أو الحقل، وبالمقابل تقوم الدالة LTRIM بمسح الفراغات عن يسار النص أو الحقل، وصيغتهما كما يلي:

```
-- Remove   Blanks From The right
SELECT RTRIM('Khalid      ');

-- Remove Blanks From the left
SELECT LTRIM('      Khalid');
```

في الحالتين معا ستكون النتيجة عبارة عن الكلمة "Khalid" من دون فراغات.

الدالتان UPPER و LOWER

تقوم هاتان الدالتان بتغيير حالة النصوص، بحيث تمكننا الدالة UPPER من تحويل أحرف النص إلى أحرف كبيرة Upper case، وبالمقابل تمكننا الدالة LOWER من تحويل حالة الأحرف إلى حالتها الصغيرة Lower case، وصيغة الدالتين كما يلي:

```
-- Change Case To Upper case
SELECT UPPER('Khalid') AS 'UPPER Case';

-- Change Case To Lower case
SELECT LOWER('Khalid') AS 'LOWER Case';
```

نتيجة الاستعلام الأول ستكون هكذا:



	UPPER Case
1	KHALID



بينما ستكون نتيجة الاستعلام الثاني هكذا:

	LOWER Case
1	khalid

الدالة CHARINDEX

تعيد لنا هذه الدالة رتبة الحرف أو النص المحدد، وصيغتها كما يلي:

```
SELECT CHARINDEX('الحرف أو النص المراد البحث عن رتبته','النص الكامل');
```

وهذا مثال لاستعمال الدالة CHARINDEX:

```
SELECT FullName,
CHARINDEX('d',FullName) AS 'Position Of "d"'
FROM Employee
```

يقوم هذا المثال بالبحث عن رتبة الحرف d في كل الأسماء، ستكون نتيجة الاستعلام على هذا الشكل:

	FullName	Position Of "d"
1	Karim Hamdi	10
2	Hamid MAKBOUL	5
3	Mohamed ALQAHTANI	7
4	Youssef Ahmed	13
5	Nihad Chawqi	5

الدالة LEN

تعيد لنا هذه الدالة طول النص أو الحقل المحدد، وصيغتها كما يلي:




```
SELECT LEN('Khalid');
```

النتيجة هي عدد الأحرف المكونة للكلمة Khalid، وهذا مثال للتوضيح:

```
SELECT FullName,
LEN(FullName) AS 'Length Of FullName'
FROM Employee
```

في هذا المثال ستكون النتيجة كما يلي:

	FullName	Length Of FullName
1	Karim Hamdi	11
2	Hamid MAKBOUL	13
3	Mohamed ALQAHTANI	17
4	Youssef Ahmed	13
5	Nihad Chawqi	12

دوال التاريخ Date Functions:

قبل أن نبدأ استعراض دوال التاريخ، سنقوم بإضافة الحقل Date_Of_Working إلى الجدول Employee، نوع هذا الحقل هو DateTime، يمكنك إضافته يدويا من خلال نافذة إضافة الحقول، أو عبر تنفيذ الأمر التالي:

```
ALTER TABLE Employee
Add Date_Of_Working DateTime
```

الدالة DATEADD

الدالة DATEADD تقوم بإضافة قيمة رقمية إلى جزء من حقل من نوع تاريخ، كإضافة بعض الأيام أو الشهور أو السنوات إلى تاريخ معين، وصيغة الدالة كما يلي:




```
DATEADD(Part_of_date, number, date)
```

المعامل الأول Part_of_date يأخذ إحدى القيم الثلاثة:

- **Year: لإجراء الإضافة على الجزء الخاص بالسنوات.**
- **Month: لإجراء الإضافة على الجزء الخاص بالشهور.**
- **Day: لإجراء الإضافة على الجزء الخاص بالأيام.**

المعامل الثاني number يأخذ القيمة الرقمية المراد إضافتها.

المعامل الثالث date هو الحقل أو القيمة التاريخية المراد إجراء الإضافة إليها.

وهذا مثال لاستعمال الدالة DATEADD على حقل من نوع تاريخي:

```
SELECT FullName,
Date_of_working as 'Date of working',
DATEADD(year,3,Date_of_working) as 'Using DATADD'
FROM Employee
```

الاستعلام أعلاه، يقوم بجلب أسماء الموظفين وتواريخ توظيفهم، بالإضافة إلى حقل ثالث يحتوي على تاريخ اشتغالهم زائد ثلاث سنوات، وعليه فإن النتيجة ستكون كما يلي:

	FullName	Date of working	Using DATADD
1	Karim Hamdi	2001-01-01 00:00:00.000	2004-01-01 00:00:00.000
2	Hamid MAKBOUL	2003-12-13 00:00:00.000	2006-12-13 00:00:00.000
3	Mohamed ALQAHTANI	1998-08-23 00:00:00.000	2001-08-23 00:00:00.000
4	Youssef Ahmed	2006-09-06 00:00:00.000	2009-09-06 00:00:00.000
5	Nihad Chawqi	2000-07-14 00:00:00.000	2003-07-14 00:00:00.000

بنفس الطريقة تستطيع إضافة الأيام والأشهر.



الدالة DATEDIFF

هذه الدالة تقوم بطرح تاريخ من تاريخ آخر، بحيث تكون القيمة الناتجة هي فارق التاريخين سواء بالأيام أو الأشهر أو السنوات، وصيغتها كما يلي:

```
DATEDIFF(Part_of_date, date1, date2)
```

المعامل الأول خاص بتحديد جزء التاريخ المراد إنجاز عملية الطرح عليه، والمعاملان الثاني والثالث هما الحقلان أو القيمتان التاريخيتان اللتان ستكونان طرفي عملية الطرح.

وهذا المثال يبين كيفية استخدام الدالة DATEDIFF:

```
SELECT DATEDIFF(Month, '12/01/2012', '12/08/2012')
```

النتيجة ستكون هي خارج عملية طرح التاريخ الأول من التاريخ الثاني بالأشهر أي أن النتيجة ستساوي 7.

الدالة DATEPART

هذه الدالة تعيد لنا جزء من قيمة تاريخية أو حقل من نوع التاريخ، وصيغتها كما يلي:

```
DATEPART(part_of_date, date)
```

المعامل الأول هو جزء التاريخ المراد استخراجه، والمعامل الثاني هو التاريخ المراد استخراج الجزء منه.

وهذا مثال لاستعمال الدالة DATEPART:




```
SELECT date_of_working AS 'Date of working',
DATEPART(YEAR, date_of_working) AS 'year',
DATEPART(MONTH, date_of_working) AS 'month',
DATEPART(DAY, date_of_working) AS 'day'
FROM Employee
```

هذا المثال يقوم بعرض التاريخ كاملا، ثم يظهر أمامه تقطيع التاريخ على شكل أيام وشهور وسنوات، وهذه صورة للنتيجة المحصل عليها:

	Date of working	year	month	day
1	2001-01-01 00:00:00.000	2001	1	1
2	2003-12-13 00:00:00.000	2003	12	13
3	1998-08-23 00:00:00.000	1998	8	23
4	2006-09-06 00:00:00.000	2006	9	6
5	2000-07-14 00:00:00.000	2000	7	14

ويوجد دوال أخرى تقوم بنفس دور الدالة DATEPART لكن على وجه التخصيص، بحيث نجد:

- الدالة MONTH تقوم باجتزاء الشهر من حقل أو قيمة تاريخية.
- الدالة DAY تقوم باجتزاء اليوم من حقل أو قيمة تاريخية.
- الدالة YEAR تقوم باجتزاء السنة من حقل أو قيمة تاريخية.

وهذا مثال جامع للدوال الثلاثة الأخيرة:

```
SELECT      MONTH(date_of_working),
            DAY(date_of_working),
            YEAR(date_of_working)
FROM Employee
```



الدالة GETDATE

تعيد لنا هذه الدالة التاريخ الحالي من نظام التشغيل، وصيغتها كما يلي:

```
SELECT  GETDATE() as 'Current Date & time'
```

النتيجة المطبوعة ستكون كما يلي:

	Current Date & time
1	2013-01-25 23:09:38.270

ولنختم مع الدوال التاريخية، سنورد مثالا أخيرا نستعمل فيه الدالة GETDATE والدالة DATEDIFF:

```
SELECT date_of_working as 'Employment date',
      getdate() as 'current date',
      datediff(year, date_of_working, getdate()) as 'Employment duration'
FROM Employee
```

في هذا المثال ستضم النتيجة ثلاثة حقول، الحقل الأول يعرض تاريخ بداية الاشتغال Employment date، والحقل الثاني سيعرض التاريخ الحالي Current date، والحقل الثالث يعرض عدد السنوات التي قضاها الموظفون في الشغل انطلاقا من تاريخ بداية الاشتغال وانتهاءً بالتاريخ الحالي، ستكون النتيجة كما يلي:

	Employment date	current date	Employment duration
1	2001-01-01 00:00:00.000	2013-01-25 23:17:40.910	12
2	2003-12-13 00:00:00.000	2013-01-25 23:17:40.910	10
3	1998-08-23 00:00:00.000	2013-01-25 23:17:40.910	15
4	2006-09-06 00:00:00.000	2013-01-25 23:17:40.910	7
5	2000-07-14 00:00:00.000	2013-01-25 23:17:40.910	13



دوال التحويل Conversion Functions:

التحويل بين أنواع البيانات من أهم العمليات التي قد تحتاجها مستقبلا، ولهذا سنورد فيما يلي بعض الدوال لتحويل البيانات من نوع إلى نوع آخر.

الدالة STR

تقوم هذه الدالة بتحويل القيم الرقمية إلى قيم نصية، وهذا مثال لاستعمالها:

```
SELECT 'The age of employee '+Fullname +' is: '+ STR(age) as 'Info'
FROM Employee
```

ستكون النتيجة كما يلي:

	Info
1	The age of employee Karim Hamdi is: 24
2	The age of employee Hamid MAKBOUL is: 23
3	The age of employee Mohamed ALQAHTANI is: 35
4	The age of employee Youssef Ahmed is: 42
5	The age of employee Nihad Chawqi is: 21

لو ألغينا دالة التحويل STR التي تقوم بتحويل قيمة العمر Age الرقمية إلى قيمة نصية، سوف نحصل على خطأ لأنه غير مسموح بدمج النصوص مع الأرقام إلا بعد القيام بعملية التحويل.

الدالة CONVERT

من مزايا هذه الدالة أنها ليست حكرا على نوع معين، بل من خلالها تستطيع تحويل أي نوع إلى نوع آخر، وهذه صيغتها:




```
CONVERT(Data_Type, value_or_column_to_convert)
```

في المعامل الأول نضع نوع البيانات المراد التحويل إليه، وفي المعامل الثاني نضع الحقل أو القيمة المراد تحويل نوعها، وهذا المثال يعرض كيفية استخدام دالة التحويل CONVERT:

```
SELECT CONVERT(int, '12') AS 'String To Int',
CONVERT(varchar, 56) AS 'Int To String'
```

السطر الأول يقوم بالتحويل من القيمة الرقمية إلى قيمة نصية، والسطر الثاني يقوم بالعكس اعتمادا على الدالة CONVERT.

الدالة CAST

دورها شبيه تماما بالدالة CONVERT، وصيغتها كما يلي:

```
CAST(value_or_column_to_convert as Data_Type)
```

بين القوسين نضع القيمة أو الحقل المراد تحويل نوعه، متبوعا بالكلمة AS ثم بنوع البيانات المراد التحويل إليه، وهذا مثال يوضح كيفية استخدام دالة التحويل CAST:

```
SELECT CAST(age AS CHAR(2)) FROM Employee
```



الفصل الرابع

تطبيقات الجبر التجريدي في لغة SQL



تذكير بالجبر التجريدي

رأينا في الفصل الأول مفاهيم الجبر التجريدي نظريا، في هذا الفصل إن شاء الله سننتقل إلى الجانب التطبيقي لهذه العمليات، على أمل أن تتوطد المعلومات النظرية في أذهاننا وتترسخ التقنيات العملية في أيادينا 😉

سنقوم بعرض تذكير موجز لكل عملية من عمليات الجبر التجريدي، مع إعطاء مثال لها بلغة SQL، والبداية ستكون مع :

الاتحاد Union (∪):

تذكير

الاتحاد Union هو علاقة تربط بين مجموعتين لهما نفس الحقول ونفس الخصائص، وتكون النتيجة عبارة عن مجموعة تضم كل عناصر المجموعتين، ويرمز لها رياضيا هكذا: R1 Ć R2

مثال عن عملية الاتحاد:

لنفترض أن لدينا جدول الموظفين وجدول رؤساء المجموعات، وهما جدولان متشابهان من حيث عدد و نوع الحقول. عملية اتحاد الجدولين ستفرز لنا جدولا ثالثا يضم كل العناصر الموجودة في الجدولين، ويمكننا التعبير عن عملية الاتحاد بلغة SQL بالشكل التالي:

```
SELECT Id, FullName FROM Employee
UNION
SELECT Id, FullName FROM Director
```



بعد تنفيذ الأمر أعلاه، سنحصل على جدول يضم أرقام وأسماء كل من الموظفين والرؤساء.

التقاطع Intersection (∩) :

تذكير

التقاطع هو ناتج ربط جدولين لهما نفس عدد الحقول، ونفس البنية، ويرمز له رياضيا بهذا الرمز R2 ∩ R1وهو يضم العناصر المشتركة بين جدولين.

مثال عن عملية التقاطع:

من نفس المثال السابق، نريد استخراج جدول يعرض لنا الأسماء المشتركة بين جدولين، وطريقة عمل ذلك كما يلي:

```
SELECT  FullName FROM Employee
INTERSECT
SELECT  FullName FROM Director
```

نتيجة الأمر أعلاه، عبارة عن جدول يضم الأسماء المشتركة بين الجدولين، مثلا لو هناك موظف اسمه "كريم" وفي جدول الرؤساء يوجد رئيس بهذا الإسم، سيظهر في النتيجة.



الاختلاف Difference (−):

> **تذكير**
>
> **الاختلاف هو الفارق الناتج عن طرح مجموعة من مجموعة أخرى، ويشترط أن يكون للمجموعتين (الجدولين) نفس البنية ونفس الحقول.**

مثال عن عملية الاختلاف:

دائما مع نفس المثال، سنقوم بجلب أسماء الموظفين الغير موجودة في جدول الرؤساء، ويمكننا الاصطلاح عليها رياضيا بهذا الشكل:

Result=Employee-Director

ترجمة هذه الصيغة إلى لغة SQL يعطينا الأمر التالي:

```
SELECT  FullName FROM Employee
WHERE FullName NOT IN
(SELECT  FullName FROM Director)
```

قم بتجريب مثال آخر من نسج أفكارك، جرب استعمال IN لوحدها لتتعرف على دور هذه الكلمة.



كما يمكننا القيام بعملية الإختلاف بكل بساطة عن طريق استخدام الكلمة Except، وصيغة استعمالها كما يلي:

```
SELECT  FullName FROM Employee
EXCEPT
SELECT  FullName FROM Director
```

سنحصل على نفس النتيجة الأولى.

الانتقاء Selection (σ):

تذكير

وتعني انتقاء بعض العناصر/الأسطر Rows من مجموعة معينة.

مثال عن عميلة الانتقاء:

هذه هي عملية Select التي رأيناها طيلة الفصول، ومن باب الإنصاف سنعطي لها مثالا أيضا ☺

تطبيق عملية الانتقاء على جدول الموظفين، تعني جلب بيانات كل الموظفين، ويمكننا كتابتها هكذا:

```
SELECT  * FROM Employee
```



الاسقاط Projection (π):

تذكير

الفرق بينه وبين الانتقاء هو كون الاسقاط يكون بغرض انتقاء الأعمدة Columns

مثال عن عملية الاسقاط:

تطبيق عملية الإسقاط على جدول الموظفين يعني إظهار بعض الحقول فقط، مثلا :

```
SELECT  FullName, Adress FROM Employee
```

الأمر أعلا سيقوم بإظهار أسماء وعناوين الموظفين فقط.

الجداء الديكارتي (×):

تذكير

ويكون الناتج عن هذه العملية عبارة عن مجموعة جديدة، تضم خارج جداء كل عنصر من المجموعتين بباقي عناصر المجموعة الأخرى.



مثال عن عملية الجداء الديكارتي:

لنفترض أن لدينا جدول للسيارات يضم أرقام وأنواع السيارات، وجدول ثان يضم خصائص السيارات، ويحتوي على اللون والثمن، كما يلي:

جدول السيارات Car:

ID	Model
1	Mercedes
2	Ferrari

جدول الخصائص Descrip:

Color	Price
Black	3 000 000
Red	7 000 000

تطبيق الجداء الديكارتي على الجدولين، يعني الحصول على جدول ثالث يضم كل الاحتمالات الممكنة، أي كما يلي:

ID	Model	Color	Price
1	Mercedes	Black	3 000 000
1	Mercedes	Red	7 000 000
2	Ferrari	Black	3 000 000
2	Ferrari	Red	7 000 000



ويمكننا إنجاز هذا الأمر في لغة SQL بالطريقة التالية:

```
SELECT Car.ID, Car.Model,
Descrip.Color, Descrip.Price
FROM Car, Descrip ORDER BY Car.ID
```

أو باستخدام الكلمة Cross Join:

```
SELECT Car.ID, Car.Model,
Descrip.Color, Descrip.Price
FROM Car
CROSS JOIN Descrip ORDER BY Car.ID
```

قم بتجريب أمثلة من ابتكارك لتتعرف أكثر على الجداء الديكارتي.

القسمة Division (÷):

تذكير

ويعني قسمة جدول على جدول آخر، بشرط أن تكون حقول الجدول الثاني متواجدة في الجدول الأول، و يرمز لها في الرياضيات ب ÷، وتكون النتيجة عبارة عن جدول يضم عناصر الجدول الأول التي تضم كل عناصر الجدول الثاني، وصيغتها الرياضية هكذا R3=R1÷R2

مثال عن عملية القسمة:

لنفترض أن لدينا جدول للمبرمجين وأرقام المشاريع التي شاركوا في إنجازها:



Developer	Project_ID
Ahmed Ismaîl	1
Younes Khalifa	2
Ahmed Ismaîl	2
Rachid Kamal	1
Younes Khalifa	1

ومن جهة أخرى لدينا جدول المشاريع:

Project_ID	Project_Description
1	Inventory System
2	Library Management

نريد عرض المبرمجين الذين شاركوا في *كل* المشاريع، للقيام بذلك سننجز عملية قسمة جدول المبرمجين على جدول المشاريع، والنتيجة هي أسماء المبرمجين الذين شاركوا في كل المشاريع.

أي أن النتيجة ستكون كما يلي:

Developer
Ahmed Ismaîl
Younes Khalifa

لإجراء عملية القسمة بواسطة SQL، يلزمنا تشغيل عقولنا جيدا، وكل حسب مقدراته الذهنية سيضع صيغة للوصول إلى النتيجة، مثلا هذا ما استطعت الوصول إليه بفضل الله وعونه:




```
SELECT Developer
FROM Developers
WHERE  project_id IN
(SELECT project_id FROM projects
WHERE  project_id IN(SELECT project_id from developers)
)
GROUP BY developer
HAVING COUNT(project_id)=(SELECT COUNT(project_id) FROM projects)
```

ربما تستطيع كتابة صيغة أخرى أسهل من هذه وذلك حسب قدرتك الفكرية، لهذا ضع نصب عينيك النتيجة، وفكر في طريقة ذكية للوصول إليها.

الربط Join (⋈):

تذكير

ويقتضي هذا النوع من العمليات جدولان لهما حقل مشترك من نفس النوع، ويستعمل بغرض البحث عن العناصر الموجودة في الجدولين من خلال تحقق شرط وجود الحقل المشترك بنفس القيمة في الجدولين، ويرمز له رياضيا بالرمز التالي ⋈

مثال عن عملية الربط:

لنفترض أن لدينا جدول المجموعات بالشكل التالي:



Group_ID	Group_Name
1	Group 1
2	Group 2
3	Group 3

ولدينا جدول الأعضاء بالشكل التالي:

Member	Group_ID
Khalid Ahmed	1
Hamid MAKBOUL	2
NABIL ZAKI	1
Saîd Morad	8

عملية الربط بين الجدولين ستتم من خلال الحقل Group_ID لأنه مشترك بينهما، لنقم بجلب أسماء الأعضاء والمجموعات معا بحيث نعرض يكون رقم المجموعات مشتركا بين الجدولين، أي أن النتيجة ابتداء ستكون بهذا الشكل:

Member	Group_Name
Khalid Ahmed	Group 1
Hamid MAKBOUL	Group 2
NABIL ZAKI	Group 1

تم جلب كل الأعضاء باستثناء، العضو الأخير لأنه ينتمي إلى المجموعة رقم 8، وهذا الرقم غير موجود في جدول المجموعات.

إذن فبإمكاننا ترجمة هذا الأمر إلى لغة SQL، كما يلي:

```
SELECT Member, Group_Name
FROM Members, Groups
WHERE Members.Group_ID=Groups.Group_ID
```



بإمكاننا أيضا إجراء عملية الربط باستخدام الكلمة JOIN، كما يعرض هذا المثال:

```
SELECT Member, Group_Name
FROM Members
JOIN Groups
ON Members.Group_ID=Groups.Group_ID
```

بعد تنفيذ هذا الاستعلام، سنحصل على نفس النتيجة السابقة.



الفصل الخامس

البرمجة في

Transact SQL



ربما قد تستغرب من اسم هذا الفصل، ولعلك تتساءل ما علاقة T-SQL بالبرمجة؟ نعم، بإمكاننا البرمجة بلغة T-SQL، تماما كأننا داخل لغة للبرمجة، سنتعرف على المتغيرات والبنيات الشرطية والتكرارية، لأنها ستفيدنا في برمجة الإجراء ات المخزنة والقوادح التي قد تستخدمها في إحدى برامجك المصممة بلغة VB.Net أو C#.Net.

المتغيرات Variables:

المتغيرات هي حيز في الذاكرة له اسم معين، ونوع بيانات، ويستخدم لتخزين قيم مؤقتة موافقة لهذا النوع، ويتم الإعلان عن المتغيرات في لغة T-SQL بالشكل التالي:

```
DECLARE @myVariable int
```

في المثال أعلاه قمنا بالإعلان عن متغيررقمي اسمه @myVariable.

إعطاء قيمة للمتغير:

لإعطاء قيمة للمتغير، هناك طريقتان.

الطريقة الأولى:

```
SET @myVariable=4
```

الطريقة الثانية:

```
SELECT @myVariable=4
```



بإمكاننا إعطاء قيمة إحدى حقول جدول ما لمتغير كما يلي:

```
SELECT @myVariable=(SELECT Age FROM Employee WHERE ID=3)
```

إظهار قيمة متغير:

لطباعة قيمة المتغير، نستخدم الكلمة PRINT، كما يلي:

```
DECLARE @myName varchar(20)

SELECT @myName='Khalid'

PRINT 'My name is: '+@myName
```

ستكون النتيجة كما يلي:

```
Messages
My name is: Khalid
```

الأمر RETURN:

يستخدم للخروج من عملية معينة، أو لإيقاف البرنامج.

البنية الشرطية IF...ELSE:

تستخدم من أجل التحقق من شرط معين، وصيغتها كما يلي:




```
IF (Conditon)
      BEGIN
      --Do Something
      END
ELSE
      BEGIN
      --Do Something
      END
```

الآن سنقوم بإنجاز مثال عملي، نستوعب من خلاله هذه المفاهيم.

```
DECLARE @Count INT

SET @Count=(SELECT COUNT(*) FROM Employee)

IF (@Count>3)

      PRINT 'Count is more then 3 Employees';

ELSE

      PRINT 'Count is less then 3 Employees';
```

قمنا بالإعلان عن متغير رقمي اسمه @Count، ثم أعطيناه عدد عناصر جدول الموظفين كقيمة، بعدها تحققنا من هذه القيمة، إن كانت أكبر من 3، أظهرنا رسالة بأن عدد عناصر جدول الموظفين أكبر من ثلاثة، وإن كان العكس أظهرنا رسالة مفادها بأن العدد أقل من 3.



البنية الشرطية باستخدام Case:

صيغتها كما يلي:

```
CASE
      WHEN Condition1 THEN Result1
      WHEN Condition2 THEN Result2
      ELSE ResultN
END
```

سنقوم بجلب بيانات الموظفين، ونتحقق من عمر كل موظف، ثم نظهر أما كل موظف رسالة حسب عمره:

```
SELECT ID, Fullname, 'Message'=CASE
When Age>20 then
'You are adult'
Else
'You are young'
End
From Employee
```

ستكون النتيجة كما يلي:

	ID	Fullname	Message
1	1	Karim Hamdi	You are young
2	2	Hamid MAKBOUL	You are adult
3	3	Mohamed ALQAHTANI	You are young
4	4	Ahmed WAHBI	You are adult
5	5	Nihad Chawqi	You are adult
6	6	Youssef Hani	You are adult



البنية التكرارية باستعمال WHILE:

تستخدم الكلمة WHILE لتكرار أمر معين مجموعة من المرات وفق العدد المحدد في الشرط،

```
WHILE Condition

      BEGIN

      --Do something

      END
```

سنعطي مثالا سهلا جدا، لكنه سيبدو لك معقدا من حيث أوامره، لهذا لا ترتبك لأننا سنشرحه جيدا إن شاء الله:

```
DECLARE @count INT

DECLARE @ID INT

SET @ID=10

SET @count=(SELECT COUNT(*) FROM employee)


WHILE (@COUNT < 9)

      BEGIN

      SET @count=(SELECT COUNT(*) FROM employee)

      INSERT INTO employee (ID, Fullname) VALUES (@ID,
      'Emp'+CONVERT(VARCHAR,@ID))

      SET @ID=@ID+1

      END
```



قمنا بالإعلان عن متغيرين رقميين، الأول count@ يساوي عدد عناصر جدول الموظفين، والثاني ID@ يساوي ابتداء 10، ثم قمنا بتكرار عملية إضافة أسطر جديدة مادام الشرط (عدد العناصر أقل من عشرة) متحققا، بمعنى أن هذا البرنامج يقوم بحساب عدد عناصر جدول الموظفين، فإن كان أقل من عشرة أضاف مجموعة من العناصر لإكمال عشرة عناصر، قم بالعودة إلى المثال وتأمله سطرا سطرا.

مثلا لو عندي في جدول الموظفين ستة عناصر، سيتم إضافة أربعة عناصر.

وهذه صورة لجدول الموظفين بعد تنفيذ البرنامج:

KHALID-PC\SQLEX...l - dbo.Employee

ID	FullName
1	Karim Hamdi
2	Hamid MAKBOUL
3	Mohamed ALQA...
4	Ahmed WAHBI
5	Nihad Chawqi
6	Youssef Hani
10	Emp10
11	Emp11
12	Emp12
13	Emp13
NULL	NULL



قم بتجريب المثال، وعدل عليه لتستوعبه جيدا.

إدارة العمليات Transactions Management:

العمليات Transactions **تمكننا من تنفيذ حزمة من الأوامر دفعة واحدة، وفي حال عدم تنفيذ أمر من هذه الأوامر يتم إلغاء تنفيذ الأوامر الأخرى، بمعنى أنها تجمع كل الأوامر على شكل أمر واحد، فإما أن تنفذ جميعا أو لا تنفذ.**

وصيغتها كما يلي:

```
BEGIN TRAN [Transaction_Name]
IF (Condition)
      --Confirm the transaction
      COMMIT TRAN [Transaction_Name]
ELSE
      --Cancel the transaction
      ROLLBACK TRAN [Transaction_Name]
```

لنفترض أن لدينا جدول المبرمجين وجدول المشاريع، لكن هذه المرة لا يوجد أي رابط بين الجدولين، ونريد حذف المبرمج رقم 100 **وكافة المشاريع التي قام ببرمجتها، الطريقة العادية كما يلي:**

```
DELETE From Programmers WHERE Programmer.Programmer_ID='100'
DELETE From Projects WHERE Projects.Programmer_ID='100'
```



لكن نفترض أنه بعد تنفيذ الأمر الأول، حدث مشكل في البرنامج، أو انقطع التيار، أو ما إلى ذلك من أسباب تحول دون تكملة باقي الأوامر، في هذه الحالة ستتبعثر البيانات، بحيث سنجد المبرمج غير موجود في جدول المبرمجين لكن رقمه حاضر في جدول المشاريع.

حالتنا هذه يسيرة، لكن افترض أنك بصدد إنجاز برنامج لإدارة البنوك ويلزم البرنامج أن يقوم باستلام الأموال من الزبون وإضافتها إلى قاعدة البيانات، ثم توقف البرنامج في مرحلة حرجة ماذا سيكون رد فعلك؟

هنا تبزغ أهمية إدارة العمليات Transactions، بحيث باستخدامها ستنفذ كل الأوامر أو تلغى كلها، وهنا نطبق إدارة العمليات على المثال السابق:

```
BEGIN TRAN
      DELETE From Programmers WHERE Programmer.Programmer_ID='100'
      DELETE From Projects WHERE Projects.Programmer_ID='100'
COMMIT TRAN
```

المررات Cursors

المررات Cursors عبارة عن كائنات تمكننا من إجراء أمر معين على مجموعة من العناصر دفعة واحدة واحدا بواحد، الأمر الذي لا نجده في طرق الإستعلام العادية، وصيغة إنشاء المررات كما يلي:




```
--الإعلان عن الممرر بإعطائه اسما و أمرا للتنفيذ
Declare Cursor_Name Cursor FOR Query
--نفتح الممرر
OPEN Cursor_Name

Fetch Cursor_Name into @Var1, @Var2

--لمعرفة نهاية الممرر
While @@fetch_status=0
BEGIN
--المرور بين العناصر واحدا واحدا
Fetch Cursor_Name into @Var1, @Var2
END
--نغلق الممرر
Close Cursor_Name
--لتحرير الموارد المستخدمة من طرف الممرر
Deallocate Cursor_Name
```

والآن تعال بنا نشرح الصيغة سطرا سطرا:

```
Declare Cursor_Name Cursor FOR Query
```

في هذا السطر نقوم بالإعلان عن الممرر مع إعطائه اسما من نوع Cursor، **ثم بعد الكلمة** FOR **نكتب الاستعلام المراد تنفيذه من طرف الممرر.**




```
OPEN Cursor_Name
```

نقوم بفتح المر ليبدأ عمله.

```
While @@fetch_status=0
```

الدالة @@fetch_status تمكننا من معرفة ما إذا وصل المرر إلى آخر عنصر أم لا، وما دامت قيمتها صفر فهذا يعني أن المرر لم يصل بعد إلى العنصر الأخير.

```
Fetch Cursor_Name into @Var1, @Var2
```

للمرور من عنصر إلى الذي يليه، استعملناه قبل الدالة وداخل الدالة، في الأول لتحديد العنصر الأول، وفي الثاني للانتقال بين العناصر.

مع العلم أن المتغيرات @Var1 و@Var2 يجب أن نعلن عنها لاستقبال القيم الناتجة عن المرر، وينبغي أن يكون نوعها وحجمها مماثلا للقيمة الناتجة، إياك أن تستعمل نفس الأسماء 😜 أنا أسميتها هكذا فقط للتوضيح.

```
Close Cursor_Name
```

هذا الأمر لإغلاق المرر بعد الانتهاء منه.

```
Deallocate Cursor_Name
```



وهذا الأمر لتحرير الموارد المستخدمة من طرف المرر.

والآن لنستوعب المررات بشكل جيد، ينبغي أن نعطي مثالا، وليكن إظهار الموظفين بالشكل التالي:

```
Employee ID : XXX,  Called : XXX,  has :XX Years.
```

هذا هو المرر ويمكنك تجريبه والتعديل عليه:

```
Declare @ID varchar(50), @FullName varchar(50), @Age int
Declare Emp_Cur Cursor FOR Select ID, FullName,Age From Employee
OPEN Emp_Cur
Fetch Emp_Cur into @ID, @Fullname,@Age
While @@fetch_status=0
BEGIN
print 'Employee ID: '+@ID+' Called: '+@FullName+' Has:
'+Convert(varchar,@Age)
Fetch Emp_Cur into @ID, @Fullname,@Age
END
Close Emp_Cur
Deallocate Emp_Cur
```



أعلنا في الأول على متغيرات بعدد ونوع الحقول التي نريد إظهارها، ثم أنشأنا ممررا لجلب البيانات من جدول الموظفين، قمنا بفتحه وعرض العناصر بواسطة Print وحولنا المتغير @Age إلى Varchar لأنه كما قلنا في أحد الفصول السابقة لا نستطيع دمج الأرقام والنصوص إلا بعد تحويل الأرقام إلى نصوص.

بعد تنفيذ هذا المر سنحصل على نتيجة كهذه:

```
Messages
Employee ID: Emp 1 Called: Khalid ESSAADANI Has: 24
Employee ID: Emp 2 Called: Hamid MAKBOUL Has: 23
Employee ID: Emp 3 Called: Mohamed ELKHAL Has: 25
Employee ID: Emp 4 Called: Younes MAADANE Has: 26
```

ملاحظة:

لم نذكر كل ما يتعلق بالممررات من باب الاختصار، وتفاديا لتكرار الشروحات لأن هنالك بعض الدوال والكلمات التي تؤدي نفس الأدوار.

الإجراءات المخزنة Stored Procedures

الإجراءات المخزنة هي مجموعة من الأوامر التي تحمل اسما، والتي بإمكاننا تنفيذها فقط باستدعاء اسمها، والجميل في الإجراء ات المخزنة أنها تتيح لنا إمكانية التنفيذ عن بعد، مثلا قاعدة البيانات التي تضم الإجراء المخزن موجودة على جهاز سيرفر، والبرنامج الذي يقوم بتنفيذها واستغلالها يوجد على جهاز عميل، هذه البنية هي التي تساعد كثيرا



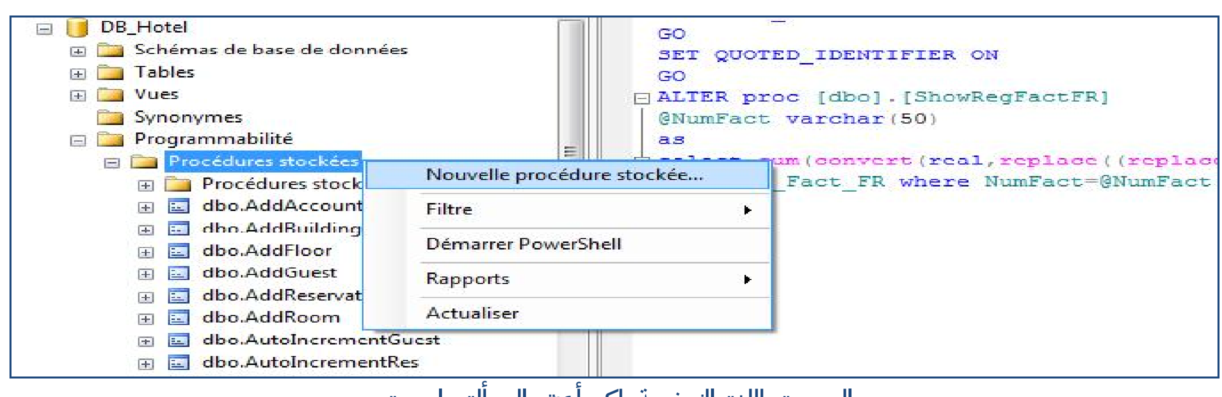
على الرفع من مستوى البرامج من نوع خادم / عميل، لأن البرنامج يقوم فقط باستلام نتيجة الإجراء المخزن، أما عمليات المعالجة والتنفيذ فكلها تتم على مستوى السيرفر الذي يكون بكفاءة عالية.

استيعاب الإجراءات المخزنة وإتقانها سيعينك كثيرا إذا كنت مبرمجا وتسعى إلى جعل برامجك تشتغل أسرع وأجود.

يمكن للإجراء المخزن أن يستقبل برامترات، وأن يجري عليها عمليات معينة، وأن يعيد لنا قيمة ناتجة عن هذه العمليات، باختصار لو عندك خلفية برمجية فالإجراء ات المخزنة شبية جدا بالدوال من حيث الدور والبنية، مع الإشارة إلى أنه يوجد مفهوم الدوال أيضا في لغة SQL وسنراها إن شاء الله مباشرة بعد الإجراء ات المخزنة.

ويتم تنفيذها محليا بواسطة الكلمة المحجوزة Execute أو Exec.

بإمكاننا إنشاء الإجراء ات المخزنة يدويا عن طريق الذهاب إلى التبويب Programmability ثم الضغط بيمين الفأرة على Stored Procedures واختيار New Stored Procedure، كما توضح الصورة التالية:



الصورة باللغة الفرنسية، لكن أعتقد المسألة واضحة.



لإنشاء إجراء مخزن، نستعمل الصيغة التالية:

```
CREATE PROC Proc_Name
--Parameters
As
//Do something
```

بحيث Proc_Name **هو اسم الإجراء المخزن.**

وفي الجزء Parameters **نقوم بالإعلان عن البرامترات المراد الاشتغال عليها داخل الإجراء، ويمكن تقسيم البرامترات إلى نوعين:**

برامترات الإدخال Input: وهي برامترات تستخدم لجلب معلومات خارجية وتطبيق بعض العمليات عليها، كما يعرض هذا المثال:

```
Create Proc Show_Employee
@ID varchar(50)
As
Select FullName, [Address], Age
From Employee
Where ID=@ID
```

البرامتر @ID من نوع Input، لأنه يتطلب إدخال قيمته لتنفيذ هذا الإجراء المخزن.

في هذا المثال سنجلب إسم وعنوان وعمر الموظف الذي رقمه الوظيفي يساوي الرقم المدخل مكان البرامتر، وطريقة تنفيذ هذا الإجراء المخزن كما يلي:




```
declare @ID varchar(50)
Set @ID='Emp 2'
Exec Show_Employee @ID
--أو بـكل بـساطة
Exec Show_Employee 'Emp 2'
```

وهناك نوع ثان من البرامترات وهو:

برامترات الإخراج Output: **ويستخدم هذا النوع من البرامترات لحفظ قيمة ناتجة عن الإجراء المخزن واستعمالها في مكان آخر، ويكون باستعمال الكلمة** Output **بعد نوع البرامتر، وهذا مثال لإجراء مخزن يستعمل برامتر للإخراج لمعرفة عدد الموظفين:**

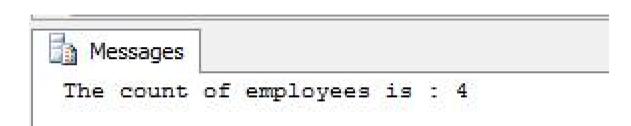
```
Create Proc Count_Employees
@Nbr int output
As
set @Nbr=(Select Count(*) From Employee)
```

لتنفيذ الإجراء أعلاه، نكتب:

```
Declare @Number int
Exec Count_Employees @Number Output
Print 'The count of employees is : ' + convert(varchar,@Number)
```

بعد التنفيذ، سنحصل على نتيجة كهذه:





وهناك نوع ثالث من الإجراءات المخزنة، يكون من نوع:

إرجاع القيمة Return: ويستعمل لإيقاف تنفيذ الإجراء المخزن بعد أمر معين، ويكون النوع المرجع دائما رقميا، وهذا مثال لاستخدام هذا النوع من الإجراء ات المخزنة، يتحقق من تطابق اسم المستخدم وكلمة المرور مع البيانات الموجودة في جدول المستخدمين، إن تحقق التطابق سيعيد لنا القيمة 1 ويوقف التنفيذ، وإن كان هناك تباين يعيد لنا 0 ويوقف البرنامج أيضا:

```
Create proc [dbo].[Login]
@Id varchar(50),@PWD varchar(50)
As
if exists(select ID from Accounts where ID=@id and [Password]=@PWD)
	begin
	return 1
	End
Else
	Begin
	return 0
	end
```



شخصيا أستعمل مثل هاته الإجراءات في برامجي عند نوافذ الدخول، أقوم فقط بمعرفة القيمة المرجعة ومن خلالها أحدد حالة الدخول أهي سليمة أم على خلاف ذلك.

أمثلة تدعيمية:

في هذا الجزء سأورد أربعة أمثلة للإجراء ات المخزنة وهي كما يلي:

إجراء مخزن يقوم بعملية الإضافة:

```
Create Proc Insert_Employee
@ID varchar(50), @FullName varchar(50), @Address text, @Age int
as
INSERT INTO [DB_Hotel].[dbo].[Employee]
           ([ID]
           ,[FullName]
           ,[Age]
           ,[Address])
     VALUES
           (@ID
           ,@FullName
           ,@Age
           ,@Address)
GO
```

إجراء مخزن يقوم بعملية التعديل:

```
Create Proc Update_Employee
@ID varchar(50), @FullName varchar(50), @Address text, @Age int
as
UPDATE [Employee]
   SET [FullName] = @FullName
      ,[Age] = @Age
      ,[Address] = @Address
 WHERE [ID] = @ID
GO
```



إجراء مخزن يقوم بعملية الحذف:

```
Create Proc Update_Employee
@ID varchar(50)
as
DELETE FROM [Employee]
      WHERE ID=@ID
GO
```

إجراء مخزن يقوم بعملية الاستعلام:

```
Create Proc Update_Employee
@ID varchar(50)
as
SELECT [FullName]
      ,[Age]
      ,[Address]
  FROM [DB_Hotel].[dbo].[Employee]
  WHERE [ID]=@ID
GO
```

تعديل الإجراءات المخزنة:

لتعديل إجراء معين، نذهب إلى التبويب الخاص بالإجراء ات المخزنة، ونضغط بيمين الفأرة على اسم الإجراء ونختار تعديل Edit، لتظهر لنا نافذة المحرر وهي تضم الإجراء المخزن المراد تعديله:

Programmabilité
Procédures stockées
Procédures stockées système
dbo.AddAccount
dbo.AddBuildir
dbo.AddFloor
dbo.AddGuest
dbo.AddReserv
dbo.AddRoom
dbo.AutoIncrer
dbo.AutoIncrer
dbo.ChangeRo

Nouvelle procédure stockée...
Modifier
Exécuter la procédure stockée...
Générer un script de la procédure stockée en tant que
Afficher les dépendances
Stratégies



كما نستطيع تعديل الإجراء بالذهاب إلى نافذة التحرير وكتابة الأمر التالي:

```
ALTER Proc Update_Employee
//Edit Content
```

حذف الإجراءات المخزنة:

لحذف إجراء مخزن، نستعمل الأمر التالي:

```
DROP PROC Proc_Name
```

بحيث Proc_Name **هو اسم الإجراء المخزن.**

أو بالضغط بيمين الفأرة على اسم الإجراء واختيار أمر الحذف Delete.



الدوال Functions:

الدالة هي مجموعة من أوامر SQL التي تحمل اسما، ويمكننا استدعاؤها من خلال هذا الإسم. وهي شبيهة بالإجراء ات المخزنة لأنها أيضا قد تستقبل برامترات، إلا أن ما يميزها هو أمر Return، لأن الدالة مجبرة على إرجاع قيمة، ويشترط في هذه القيمة أن تكون رقمية أو عبارة عن جدول Table.

وصيغة إنشاء الدوال كما يلي:

```
CREATE FUNCTION Function_Name(
@Parameters
)
RETURNS Data_Type
AS
	Begin
	//Do Something
	RETURN Value
	END
```

نعطي الدالة اسما، ثم نقوم بتحديد البرامترات - إن وجدت -، الكلمة RETURNS نستخدمها لتحديد نوع الدالة، بعد الكلمة BEGIN نكتب الأوامر المراد تنفيذها، ثم في الأخير نعيد القيمة الناتجة عن الأوامر، ويلزمها أن تكون من نفس النوع الذي حددناه أولا بعد RETURNS.



الآن سننشئ دالة تقوم بحساب عدد المشاريع التي شارك فيها المبرمج الذي نمرر رقمه في البرامتر:

```
CREATE FUNCTION Count_Projects(@ID VARCHAR(50))
RETURNS INT
AS
BEGIN
DECLARE @Nbr INT
SELECT @Nbr=(SELECT Count(*) FROM Projects WHERE ProgrammerID=@ID)
RETURN @Nbr
END
```

الدالة تستقبل الرقم الوظيفي للمبرمج، ثم تعيد لنا قيمة رقمية تضم عدد المشاريع التي شارك فيها هذا المبرمج، لمشاهدة نتيجة هذه الدالة، نقوم فقط بكتابة اسمها مع تعويض البرامتر بقيمة من نفس نوعه، كما يظهر هذا المثال:

```
SELECT [dbo].[Count_Projects] ('Dev 1')
```

بحيث `[Count_Projects]` هو اسم الدالة، و `[dbo]` اسم المستخدم لقاعدة البيانات التي تضم هذه الدالة.

بعد تنفيذ السطر أعلاه، ستحصل على نتيجة كهذه:

Résultats | Messages

	(Aucun nom de colonne)
1	4



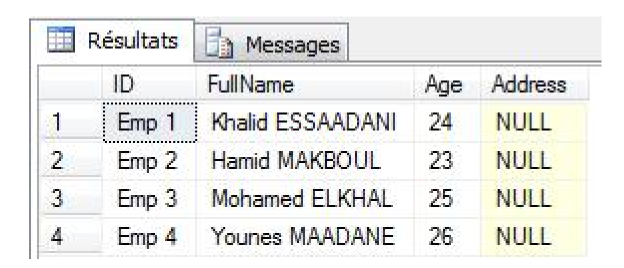
الآن سنجرب مثالا آخر يعيد لنا جدولا وليس قيمة، هذه المرة سنجلب لائحة الموظفين الذين يفوق سنهم العمر الممرر عبر البرامتر:

```
CREATE FUNCTION Employees_By_Age(@Age INT)
RETURNS TABLE
AS
RETURN (SELECT * FROM Employee
WHERE Age >@Age)
```

استدعاء هذه الدالة كما يلي:

```
SELECT * FROM [dbo].[Employees_By_Age] (20)
```

بعد تنفيذ السطر أعلاه، ستحصل على نتيجة مماثلة لما يلي:



Résultats | Messages

	ID	FullName	Age	Address
1	Emp 1	Khalid ESSAADANI	24	NULL
2	Emp 2	Hamid MAKBOUL	23	NULL
3	Emp 3	Mohamed ELKHAL	25	NULL
4	Emp 4	Younes MAADANE	26	NULL



تعديل الدوال:

التعديل شبيه جدا بالإجراءات المخزنة، يكون باستخدام الكلمة ALTER:

```
ALTER FUNCTION Function_Name(
@Parameters
)
RETURNS Data_Type

-- Do Somthing
```

حذف الدوال:

عملية الحذف أيضا تكون باستخدام الكلمة DROP مثل باقي كائنات قاعدة البيانات:

```
DROP FUNCTION Function_Name
```

القوادح Triggers:

القوادح هي كائنات نبرمجها لتؤدي دور مراقبة البيانات، وأيضا التأثير على مسار العمليات، بحيث من خلالها نستطيع التحكم في عمليات الإضافة والحذف والتعديل دون أي تدخل منا، ويتم ربط القوادح بالجداول للتحكم في العمليات الممكن القيام بها على الجداول، ويوجد نوعان من القوادح:

- **القوادح INSTEAD OF: يتم تنفيذها قبل عمليات الإضافة والتعديل والحذف.**



القواذح AFTER أو FOR: يتم تنفيذها بعد عمليات الإضافة والتعديل والحذف.

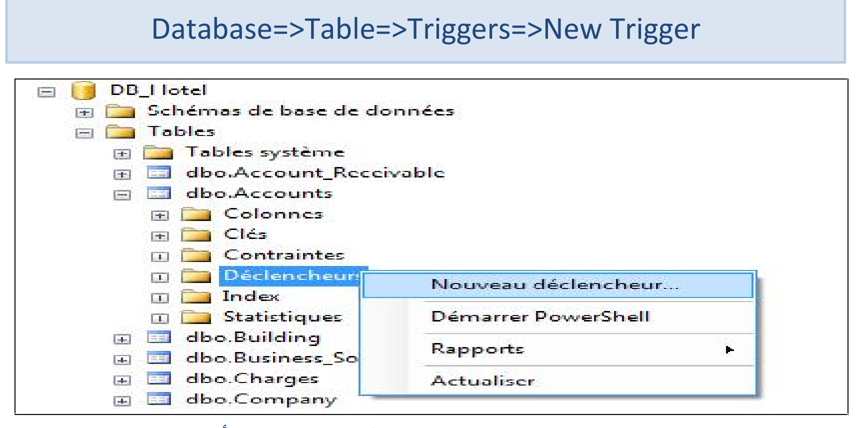
حتى نفهم كيف تمر عمليات الإضافة والتعديل والحذف ينبغي أن نفهم أولا، أنه عند إضافة سطر جديد إلى جدول ما، فإن الإضافة تتم أولا في جدول مؤقت يسمى Inserted، حتى يتم تفعيلها، بعد ذلك تظهر في الجدول الحقيقي، أيضا عملية الحذف، فإنك حينما تقوم بحذف سطر ما فإنه لا يحذف فعليا وإنما يتم نقله إلى جدول مؤقت اسمه Deleted إلى حين تفعيل الحذف بمعنى عدم وجود قادح أو أمر معين لإلغاء عملية الحذف.

أما عملية التحديث فهي تتطلب الجدولين معا Inserted و Deleted، بحيث يتم نقل القيم القديمة إلى الجدول Deleted ويتم وضع القيم الجديدة في الجدول Inserted.

إنشاء القوادح Triggers:

لإنشاء قادح جديد على جدول معين، نقوم بالضغط على اسم الجدول فتظهر لنا مجموعة من الخيارات من ضمنها Triggers، نضغط عليها بيمين الفأرة ونختار New Trigger:

Database=>Table=>Triggers=>New Trigger



إذا كانت عندك نسخة انجليزية اتبع الخطوات المكتوبة أعلى الصورة



سيعطيك الصيغة العامة لكتابة القوادح بالإضافة إلى مجموعة من التعاليق:

```
--
-- Use the Specify Values for Template Parameters
-- command (Ctrl-Shift-M) to fill in the parameter
-- values below.
--
-- See additional Create Trigger templates for more
-- examples of different Trigger statements.
--
-- This block of comments will not be included in
-- the definition of the function.
-- ================================================
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:      <Author,,Name>
-- Create date: <Create Date,,>
-- Description: <Description,,>
-- =============================================
CREATE TRIGGER <Schema_Name, sysname, Schema_Name>.<Trigger_Name, sysname, Trigger_Name>
   ON  <Schema_Name, sysname, Schema_Name>.<Table_Name, sysname, Table_Name>
   AFTER <Data_Modification_Statements, , INSERT,DELETE,UPDATE>
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
```

قم بحذف المحتوى واكتب فيه أوامر القادح المراد إنشاؤه، صيغة إنشاء القوادح باختصار كما يلي:

```
CREATE TRIGGER Trigger_Name
ON Table_Name
AFTER | FOR | Instead OF     DELETE, Insert, Update
AS
--Do Something
```

يمكنك اختيار نوع القادح كما تريد، ونوع العملية المراد التعامل معها.



يمكن للقادح الواحد أن يتحكم في أكثر من عملية، أي بإمكاننا تحديد الإضافة والتعديل والحذف مرة واحدة إذا أردنا ذلك.

والآن تعال بنا نتأمل هذا القادح البسيط الذي يمنع عملية الحذف من جدول الموظفين:

```
CREATE TRIGGER TR_Delete

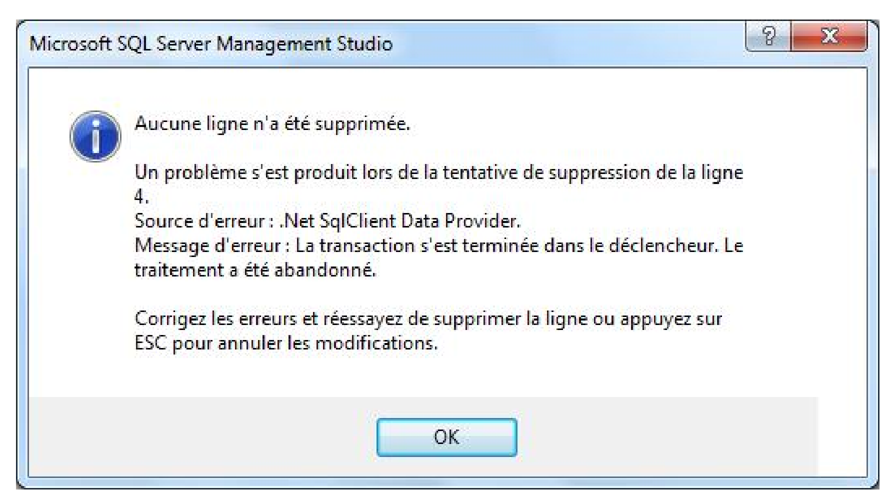
ON Employee

FOR DELETE

AS

ROLLBACK
```

الآن لو أردنا حذف أي عنصر من جدول الموظفين ستطالعنا رسالة الخطأ التالية:



الرسالة بالفرنسية مفادها أن عملية الحذف غير ممكنة بسبب مشكل ما، بإمكاننا إظهار نص خاص بنا داخل هذه الرسالة بواسطة الأمر Raiserror.



الأمر Raiserror :

الأمر Raiserror **يمكننا من إظهار رسالة خطأ للمستخدم، وصيغتها المختصرة كما يلي:**

```
RAISERROR('Text_Message',Level_Message,State_Message)
```

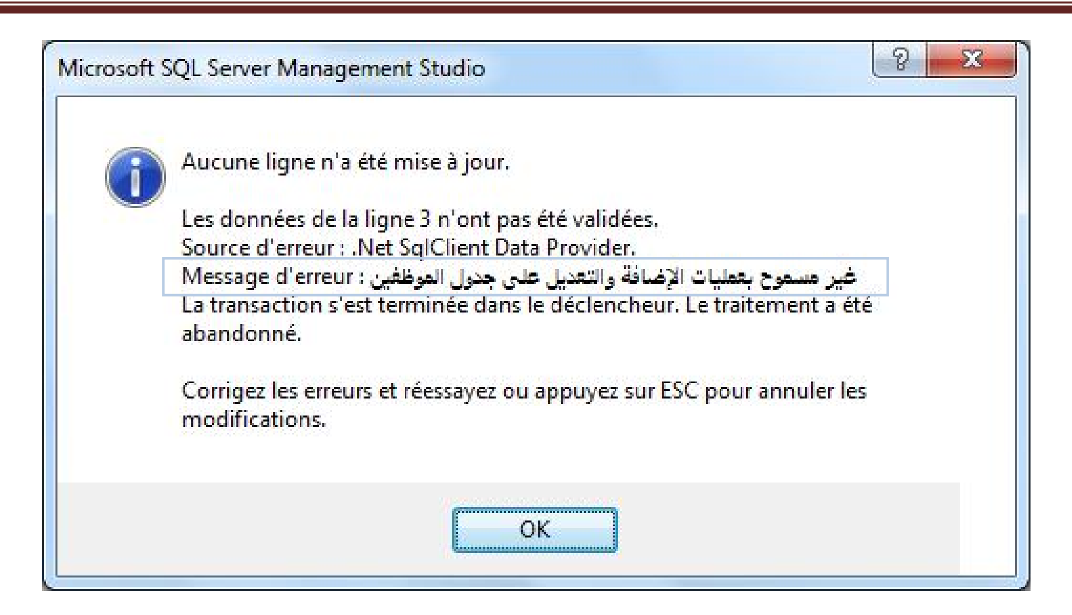
بحيث Text_Message **هو نص رسالة الخطأ، و** Level_Message **هو درجة الخطأ ويأخذ قيمة رقمية بين** 0 **و** 24، **أما** State_Message **فهي حالة الخطأ وهي أيضا تأخذ قيمة رقمية محصورة بين** 1 **و** 127.

الآن تعال بنا ننشىء قادحا جديدا يقوم بمنع عملية التحديث والإضافة في جدول الموظفين، ويظهر رسالة للمستخدم تحتوي نصا تحذيريا بأن عمليتي الإضافة والتعديل غير مسموح بهما على جدول الموظفين:

```
CREATE TRIGGER TR_Insert_Update
ON Employee
FOR INSERT, UPDATE
AS
RAISERROR('غير مسموح بعمليات الإضافة و التعديل على جدول الموظفين ',12,3);
ROLLBACK
```

بعد تنفيذ هذا القادح، وذهابك إلى جدول الموظفين للقيام بإضافة سطر جديد أو تعديل سطر موجود، ستطالعك لرسالة التالية:





حذف القوادح

لحذف قادح معين، يمكنك الذهاب إلى التبويب Triggers الموجود ضمن تبويبات الجدول المعني، وقم بحذفه بكل بساطة، أو بواسطة تنفيذ الأمر التالي:

```
DROP TRIGGER Trigger_Name
```

تعديل القوادح

عملية التعديل أيضا بنفس الطريقة، اذهب إلى القادح واختر أمر التعديل Edit أو استعمل الكلمة ALTER كما رأينا مع باقي الكائنات سابقا.



الخاتمة

تم بفضل الله وعونه استعراض أغلب مفاهيم لغة SQL، بطريقة لم أجد أيسر منها في الشرح والتبسيط، ومع ذلك فقد أكون نسيت مفهوما ما، أو انتكست في شرح جزء معين، لذا لو عند حضراتكم أية ملاحظات أو اقتراحات أو مؤاخذات أو أسئلة فلا تترددوا في مراسلتي عبر بريدي الالكتروني، وإن شاء الله سأرد عليكم قدر المستطاع.

Khalid_ESSAADANI@Hotmail.FR

للتواصل المباشر مع صاحب الكتاب، التحقوا بنا على صفحة خطوة إلى الأمام:

https://www.facebook.com/Khotwa.Amam

دام لكم البشر والفرح والسلام عليكم ورحمة الله وبركاته